

Silent Transitions in Automata with Storage

Georg Zetzsche

February 18, 2013

We consider the computational power of silent transitions in one-way automata with storage. Specifically, we ask which storage mechanisms admit a transformation of a given automaton into one that accepts the same language and reads at least one input symbol in each step.

We study this question using the model of valence automata. Here, a finite automaton is equipped with a storage mechanism that is given by a monoid.

This work presents generalizations of known results on silent transitions. For two classes of monoids, it provides characterizations of those monoids that allow the removal of λ -transitions. Both classes are defined by graph products of copies of the bicyclic monoid and the group of integers. The first class contains pushdown storages as well as the blind counters while the second class contains the blind and the partially blind counters.

1 Introduction

We consider the problem of removing silent transitions from one-way automata with various kinds of storage. Specifically, we ask for which kinds of storage the real-time and the general version have equal computational power.

This is an interesting problem for two reasons. First, it has consequences for the time and space complexity of the membership problem for these automata. For automata with silent transitions, it is not even clear whether the membership problem is decidable. If, however, an automaton has no silent transitions, we only have to consider paths that are at most as long as the word at hand. In particular, if we can decide whether a sequence of storage operations is valid using linear space, we can also solve the membership problem (nondeterministically) with a linear space bound. Similarly, if we can decide validity of such a sequence in polynomial time, we can solve the membership problem in (nondeterministic) polynomial time.

Second, we can interpret the problem as a question on resource consumption of restricted machine models: we ask for which storage mechanisms we can process every input word by executing only a bounded number of operations per symbol.

There is a wide variety of machine models that consist of a finite state control with a one-way input and some mechanism to store data, for example (higher order) pushdown automata, various kinds of counter automata [12], or off-line Turing machines that can only move right on the input tape.

For some of these models, it is known whether λ -transitions can be eliminated. For example, the Greibach normal form allows their removal from *pushdown automata* [11]. Furthermore, for *blind counter automata* (i.e., the counters can go below zero and a zero-test is only performed in the end), Greibach also has also shown that λ -transitions can be avoided [12]. However, for *partially blind counter automata* (i.e., the counters cannot go below zero and are only zero-tested in the end) or, equivalently, Petri nets, there are languages for which λ -transitions are indeed necessary [12, 16, 17].

The aim of this work is to generalize these results and obtain insights into how the properties of the storage mechanism influence the computational power of the real-time variant.

In order to study the expressive power of real-time computations in greater generality, we use the model of *valence automata*. For our purposes, a storage mechanism consists of a (possibly infinite) set of states and partial transformations operating on them. Such a mechanism often works in a way such that a computation is considered valid if the composition of the applied transformations is the identity. For example, in a pushdown storage, the operations *push* and *pop* (for each participating stack symbol) and compositions thereof are partial transformations on the set of words over some alphabet. In this case, a computation is valid if, in the end, the stack is brought back to the initial state, i.e., the identity transformation has been applied. Furthermore, in a partially blind counter automaton, a computation is valid if it leaves the counters with value zero, i.e., the composition of the applied operations *increase* and *decrease* is the identity. Therefore, the set of all compositions of the partial transformations forms a monoid such that in many cases, a computation is valid if the composition of the transformations is the identity map.

A valence automaton is a finite automaton in which each edge carries, in addition to an input word, an element of a monoid. A word is then accepted if there is a computation that spells the word and for which the product of the monoid elements is the identity. Valence automata have been studied throughout the last decades [5–7, 14, 15, 18, 20–22].

The contribution of this work is threefold. On the one hand, we introduce a class of monoids that accommodates, among others, all storage mechanisms for which we mentioned previous results on silent transitions. The monoids in this class are graph products of copies of the bicyclic monoid and the integers. On the other hand, we present two generalizations of those established facts. Our first main result is a characterization of those monoids in a certain subclass for which λ -transitions can be eliminated. This subclass contains, among others, both the monoids corresponding to pushdown storages as well as those corresponding to blind multicounter storages. Thus, we obtain a generalization and unification of two of the three λ -removal results above. For those storage mechanisms in this subclass for which we can remove λ -transitions, there is a simple intuitive description.

The second main result is a characterization of the previous kind for the class of those

storage mechanisms that consist of a number of blind counters and a number of partially blind counters. Specifically, we show that we can remove λ -transitions if and only if there is at most one partially blind counter. Again, this generalizes and unifies two of the three results above.

The rest of the paper is organized as follows. In Section 2, we will fix notation and define some basic concepts. In Section 3, we state the main results, describe how they relate to what is known, and explain key ideas. Sections 4, 5, and 6 contain auxiliary results needed in Section 7, which presents the proofs of the main results.

2 Basic Notions

We assume that the reader has some basic knowledge on formal languages and monoids. In this section, we will fix some notation and introduce basic concepts.

A *monoid* is a set M together with an associative operation and a neutral element. Unless defined otherwise, we will denote the neutral element of a monoid by 1 and its operation by juxtaposition. That is, for a monoid M and $a, b \in M$, $ab \in M$ is their product. For $a, b \in M$, we write $a \sqsubseteq b$ if there is a $c \in M$ such that $b = ac$. By **1**, we denote the trivial monoid that consists of just one element.

We call a monoid *commutative* if $ab = ba$ for any $a, b \in M$. A subset $N \subseteq M$ is said to be a *submonoid* of M if $1 \in N$ and $a, b \in N$ implies $ab \in N$. For a subset $N \subseteq M$, let $\langle N \rangle$ be the intersection of all submonoids N' of M that contain N . That is, $\langle N \rangle$ is the smallest submonoid of M that contains N . $\langle N \rangle$ is also called the *submonoid generated by N* . In each monoid M , we have the following submonoids:

$$\begin{aligned} \mathbf{H}(M) &= \{a \in M \mid \exists b \in M : ab = ba = 1\}, \\ \mathbf{R}(M) &= \{a \in M \mid \exists b \in M : ab = 1\}, \\ \mathbf{L}(M) &= \{a \in M \mid \exists b \in M : ba = 1\}. \end{aligned}$$

When using a monoid M as part of a control mechanism, the subset

$$\mathbf{J}(M) = \{a \in M \mid \exists b, c \in M : bac = 1\}$$

will play an important role. By M^n , we denote the n -fold direct product of M , i.e. $M^n = M \times \cdots \times M$ with n factors.

Let $S \subseteq M$ be a subset. If there is no danger of confusion with the n -fold direct product, we write S^n for the set of all elements of M that can be written as a product of n factors from S .

Let Σ be a fixed countable set of abstract symbols, the finite subsets of which are called *alphabets*. For an alphabet X , we will write X^* for the set of words over X . The empty word is denoted by $\lambda \in X^*$. Together with the concatenation as its operation, X^* is a monoid. We will regard every $x \in X$ as an element of X^* , namely the word consisting of only one occurrence of x . For a symbol $x \in X$ and a word $w \in X^*$, let $|w|_x$ be the number of occurrences of x in w . For a subset $Y \subseteq X$, let $|w|_Y = \sum_{x \in Y} |w|_x$. By $|w|$, we will refer to the length of w . Given an alphabet X and a monoid M , subsets

of X^* and $X^* \times M$ are called *languages* and *transductions*, respectively. A *family* is a set of languages that is closed under isomorphism and contains at least one non-trivial member. For a subset $Y \subseteq X$, we define the homomorphism $\pi_Y : X^* \rightarrow Y^*$ by $\pi_Y(y) = y$ for $y \in Y$ and $\pi_Y(x) = \lambda$ for $x \in X \setminus Y$.

Given an alphabet X , we write X^\oplus for the set of maps $\alpha : X \rightarrow \mathbb{N}$. Elements of X^\oplus are called *multisets*. By way of pointwise addition, written $\alpha + \beta$, X^\oplus is a commutative monoid (also called the *free commutative monoid* over X). We write 0 for the empty multiset, i.e. the one that maps every $x \in X$ to $0 \in \mathbb{N}$. For $\alpha \in X^\oplus$, let $|\alpha| = \sum_{x \in X} \alpha(x)$. The *Parikh mapping* is the mapping $\Psi : \Sigma^* \rightarrow \Sigma^\oplus$ defined by $\Psi(w)(x) = |w|_x$ for all $w \in \Sigma^*$ and $x \in \Sigma$.

Let A be a (not necessarily finite) set of symbols and $R \subseteq A^* \times A^*$. The pair (A, R) is called a (*monoid*) *presentation*. The smallest congruence of A^* containing R is denoted by \equiv_R and we will write $[w]_R$ for the congruence class of $w \in A^*$. The *monoid presented by* (A, R) is defined as A^*/\equiv_R . Note that since we did not impose a finiteness restriction on A , every monoid has a presentation. Furthermore, for monoids M_1, M_2 we can find presentations (A_1, R_1) and (A_2, R_2) such that $A_1 \cap A_2 = \emptyset$. We define the *free product* $M_1 * M_2$ to be presented by $(A_1 \cup A_2, R_1 \cup R_2)$. Note that $M_1 * M_2$ is well-defined up to isomorphism. By way of the injective morphisms $[w]_{R_i} \mapsto [w]_{R_1 \cup R_2}$, $w \in A_i^*$ for $i = 1, 2$, we will regard M_1 and M_2 as subsets of $M_1 * M_2$. In analogy to the n -fold direct product, we write $M^{(n)}$ for the n -fold free product of M .

Rational Sets Let M be a monoid. An *automaton over* M is a tuple $A = (Q, M, E, q_0, F)$, in which Q is a finite set of *states*, E is a finite subset of $Q \times M \times Q$ called the set of *edges*, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. The *step relation* \Rightarrow_A of A is a binary relation on $Q \times M$, for which $(p, a) \Rightarrow_A (q, b)$ iff there is an edge (p, c, q) such that $b = ac$. The set generated by A is then

$$S(A) = \{a \in M \mid \exists q \in F : (q_0, 1) \Rightarrow_A^* (q, a)\}.$$

A set $R \subseteq M$ is called *rational* if it can be written as $R = S(A)$ for some automaton A over M . The set of rational subsets of M is denoted by $\text{RAT}(M)$. Given two subsets $S, T \subseteq M$, we define $ST = \{st \mid s \in S, t \in T\}$. Since $\{1\} \in \text{RAT}(M)$ and $ST \in \text{RAT}(M)$ whenever $S, T \in \text{RAT}(M)$, this operation makes $\text{RAT}(M)$ a monoid itself.

Let C be a commutative monoid for which we write the composition additively. For $n \in \mathbb{N}$ and $c \in C$, we use nc to denote $c + \dots + c$ (n summands). A subset $S \subseteq C$ is *linear* if there are elements s_0, \dots, s_n such that $S = \{s_0 + \sum_{i=1}^n a_i s_i \mid a_i \in \mathbb{N}, 1 \leq i \leq n\}$. A set $S \subseteq C$ is called *semilinear* if it is a finite union of linear sets. By $\text{SL}(C)$, we denote the set of semilinear subsets of C . It is well-known that $\text{RAT}(C) = \text{SL}(C)$ for commutative C (we will, however, sometimes still use $\text{SL}(C)$ to make explicit that the sets at hand are semilinear). Moreover, $\text{SL}(C)$ is a commutative monoid by way of the product $(S, T) \mapsto S + T = \{s + t \mid s \in S, t \in T\}$. It is well-known that the class of semilinear subsets of a free commutative monoid is closed under Boolean operations (see [8, 9]).

In slight abuse of terminology, we will sometimes call a language L semilinear if the set $\Psi(L)$ is semilinear. If there is no danger of confusion, we will write S^\oplus instead of

$\langle S \rangle$ if S is a subset of a commutative monoid C . Note that, if X is regarded as a subset of the set of multisets over X , the two meanings of X^\oplus coincide.

Valence Automata A *valence automaton* over M is an automaton A over $X^* \times M$, where X is an alphabet. An edge (p, w, m, q) in A is called a λ -transition if $w = \lambda$. A is called λ -free if it has no λ -transitions. The *language accepted by* A is defined as

$$L(A) = \{w \in X^* \mid (w, 1) \in S(A)\}.$$

The class of languages accepted by valence automata and λ -free valence automata over M is denoted by $\mathbf{VA}^\lambda(M)$ and $\mathbf{VA}(M)$, respectively.

A *finite automaton* is a valence automaton over the trivial monoid $\mathbf{1}$. For a finite automaton $A = (Q, X^* \times \mathbf{1}, E, q_0, F)$, we also write $A = (Q, X, E, q_0, F)$. Languages accepted by finite automata are called *regular languages*. The finite automaton A is *spelling*, if $E \subseteq Q \times X \times Q$, i.e. every edges carries exactly one letter. Let M and C be monoids. A *valence transducer over M with output in C* is an automaton A over $X^* \times M \times C$, where X is an alphabet. The *transduction performed by* A is

$$T(A) = \{(x, c) \in X^* \times C \mid (x, 1, c) \in S(A)\}.$$

A valence transducer is called λ -free if it is λ -free as a valence automaton. We denote the class of transductions performed by (λ -free) valence transducers over M with output in C by $\mathbf{VT}^\lambda(M, C)$ ($\mathbf{VT}(M, C)$).

Graphs A *graph* is a pair $\Gamma = (V, E)$ where V is a finite set and $E \subseteq \{S \subseteq V \mid |S| \leq 2\}$. The elements of V are called *vertices* and those of E are called *edges*. If $\{v\} \in E$ for some $v \in V$, then v is called a *looped* vertex and otherwise it is *unlooped*. A *subgraph* of Γ is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq E$. Such a subgraph is called *induced (by V')* if $E' = \{S \in E \mid S \subseteq V'\}$, i.e. E' contains all edges from E incident to vertices in V' . By $\Gamma \setminus \{v\}$, for $v \in V$, we denote the subgraph of Γ induced by $V \setminus \{v\}$. Given a graph $\Gamma = (V, E)$, its *underlying loop-free graph* is $\Gamma' = (V, E')$ with $E' = E \cap \{S \subseteq V \mid |S| = 2\}$. For a vertex $v \in V$, the elements of $N(v) = \{w \in V \mid \{v, w\} \in E\}$ are called *neighbors* of v . A *looped clique* is a graph in which $E = \{S \subseteq V \mid |S| \leq 2\}$. Moreover, a *clique* is a loop-free graph in which any two distinct vertices are adjacent. Finally, an *anti-clique* is a graph with $E = \emptyset$.

A presentation (A, R) in which A is a finite alphabet is a *Thue system*. To each graph $\Gamma = (V, E)$, we associate the Thue system $T_\Gamma = (X_\Gamma, R_\Gamma)$ over the alphabet $X_\Gamma = \{a_v, \bar{a}_v \mid v \in V\}$. R_Γ is defined as

$$R_\Gamma = \{(a_v \bar{a}_v, \lambda) \mid v \in V\} \cup \{(xy, yx) \mid x \in \{a_v, \bar{a}_v\}, y \in \{a_w, \bar{a}_w\}, \{v, w\} \in E\}.$$

In particular, we have $(a_v \bar{a}_v, \bar{a}_v a_v) \in R_\Gamma$ whenever $\{v\} \in E$. To simplify notation, the congruence \equiv_{T_Γ} is then also denoted by \equiv_Γ . In order to describe the monoids we use to model storage mechanisms, we define monoids using graphs. To each graph Γ , we associate the monoid

$$\mathbf{M}\Gamma = X_\Gamma^* / \equiv_\Gamma.$$




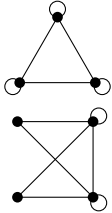
Graph Γ	Monoid $\mathbb{M}\Gamma$	Storage mechanism
	$\mathbb{B}^{(3)}$	Pushdown (with three symbols)
	\mathbb{B}^3	Three partially blind counters
	\mathbb{Z}^3	Three blind counters
	$\mathbb{B}^{(2)} \times \mathbb{Z}^2$	Pushdown (with two symbols) and two blind counters

Table 1: Examples of storage mechanisms

If Γ consists of one vertex and has no edges, $\mathbb{M}\Gamma$ is also denoted as \mathbb{B} and we will refer to it as the *bicyclic monoid*. The generators a_v and \bar{a}_v are then also written a and \bar{a} , respectively.

3 Results

Storage mechanisms as monoids First of all, we will see how pushdown storages and (partially) blind counters can be regarded as monoids of the form $\mathbb{M}\Gamma$. See Table 1 for a set of examples. It is not hard to see that in the bicyclic monoid \mathbb{B} , a word over the generators a and \bar{a} is the identity if and only if in every prefix of the word, there are at least as many a 's as there are \bar{a} 's and in the whole word, there are as many a 's as there are \bar{a} 's. Thus, a valence automaton over \mathbb{B} is an automaton with one counter that cannot go below zero and is zero in the end. Here, the increment operation corresponds to a and the decrement corresponds to \bar{a} .

Observe that building the direct product means that both storage mechanisms (described by the factors) are available and can be used simultaneously. Thus, valence automata over \mathbb{B}^n are automata with n partially blind counters. Therefore, if Γ is a clique, then $\mathbb{M}\Gamma \cong \mathbb{B}^n$ corresponds to a partially blind multicounter storage.

Furthermore, the free product of a monoid M with \mathbb{B} yields what can be seen as a stack of elements of M : a valence automaton over $M * \mathbb{B}$ can store a sequence of elements of M (separated by a) such that it can only remove the topmost element if it is the identity element. The available operations are those available for M (which then operate on the topmost entry) and in addition *push* (represented by a) and *pop* (represented by \bar{a}). Thus, $\mathbb{B} * \mathbb{B}$ corresponds to a stack over two symbols. In particular, if Γ is an anti-clique (with at least two vertices), then $\mathbb{M}\Gamma \cong \mathbb{B}^{(n)}$ represents a pushdown storage.


Finally, valence automata over \mathbb{Z}^n (regarded as a monoid by way of addition) correspond to automata with n blind counters. Hence, if Γ is a looped clique, then $\mathbb{M}\Gamma \cong \mathbb{Z}^n$ corresponds to a blind multicounter storage.

Main results Our class of monoids that generalizes pushdown and blind multicounter storages is the class of $\mathbb{M}\Gamma$ where in Γ , any two looped vertices are adjacent and any two unlooped vertices are not adjacent. Our first main result is the following.

Theorem 3.1. *Let Γ be a graph such that*

- *between any two looped vertices, there is an edge, and*
- *between any two unlooped vertices, there is no edge.*

Then the following assertions are equivalent:

- (1) $\text{VA}(\mathbb{M}\Gamma) = \text{VA}^\lambda(\mathbb{M}\Gamma)$.
- (2) *Every language in $\text{VA}^\lambda(\mathbb{M}\Gamma)$ is context-sensitive.*
- (3) *The membership problem of each language in $\text{VA}^\lambda(\mathbb{M}\Gamma)$ is in NP.*
- (4) *Every language in $\text{VA}^\lambda(\mathbb{M}\Gamma)$ is decidable.*
- (5) Γ *does not contain*  *as an induced subgraph.*

Note that this generalizes the facts that in pushdown automata and in blind counter automata, λ -transitions can be avoided.

It turns out that the storages that satisfy the equivalent conditions of Theorem 3.1 (and the hypothesis), are exactly those in the following class.

Definition 3.2. Let \mathcal{C} be the smallest class of monoids such that $\mathbf{1} \in \mathcal{C}$ and whenever $M \in \mathcal{C}$, we also have $M \times \mathbb{Z} \in \mathcal{C}$ and $M * \mathbb{B} \in \mathcal{C}$.

Thus, \mathcal{C} contains those storage types obtained by successively *adding blind counters* and *building a stack of elements*. For example, we could have a stack each of whose entries contains n blind counters. Or we could have an ordinary pushdown and a number of blind counters. Or a stack of elements, each of which is a pushdown storage and a blind counter, etc.

Our second main result concerns storages consisting of a number of blind counters and a number of partially blind counters.

Theorem 3.3. *Let Γ be a graph such that between any two distinct vertices, there is an edge. Then*

$$\text{VA}(\mathbb{M}\Gamma) = \text{VA}^\lambda(\mathbb{M}\Gamma) \text{ if and only if } r \leq 1,$$

where r is the number of unlooped vertices in Γ .

In other words, when you have r partially blind counters and s blind counters, λ -transitions can be eliminated if and only if $r \leq 1$. Note that this generalizes Greibach's result that in partially blind multicounter automata, λ -transitions are indispensable.

Key technical ingredients As a first step, we show that for $M \in \mathcal{C}$, all languages in $\text{VA}^\lambda(M)$ are semilinear. This is needed in various situations throughout the proof. We prove this using an old result by van Leeuwen [23], which says that languages that are algebraic over a class of semilinear languages are semilinear themselves. Thereby, the corresponding lemma 4.4 slightly generalizes one of the central components in a decidability result by Lohrey and Steinberg on the rational subset membership problem for graph groups [19] and provides a simpler proof (relying, however, on van Leeuwen’s result).

Second, we use an undecidability result by Lohrey and Steinberg [19] concerning the rational subset membership problem for certain graph groups. We deduce that for monoids M outside of \mathcal{C} (and satisfying the hypothesis of Theorem 3.1), $\text{VA}^\lambda(M)$ contains undecidable languages.

Third, in order to prove our claim by induction on the construction of $M \in \mathcal{C}$, we use a significantly stronger induction hypothesis: we show that it is not only possible to remove λ -transitions from valence automata, but also from valence transducers with output in a commutative monoid. Here, however, the constructed valence transducer is allowed to output a semilinear set in each step. Monoids that admit such a transformation will be called *strongly λ -independent*.

Fourth, we develop a normal form result for rational subsets of monoids in \mathcal{C} (see section 6). Such normal form results have been available for monoids described by monadic rewriting systems (see, for example, [1]), which was applied by Render and Kambites to monoids representing pushdown storages [22]. Under different terms, this normal form trick has been used by Bouajjani, Esparza, and Maler [2] and by Caucal [3] to describe rational sets of pushdown operations. However, since the monoids in \mathcal{C} allow commutation of certain non-trivial elements, a more general technique was necessary here. In the case of monadic rewriting systems, one transforms a finite automaton according to rewriting rules by gluing in new edges. Here, we glue in automata accepting sets that are semilinear by earlier steps in the proof. See Lemma 6.2 for details.

Fifth, we have three techniques to eliminate λ -transitions from valence transducers while retaining the output in a commutative monoid. Here, we need one technique to show that if M is strongly λ -independent, then $M \times \mathbb{Z}$ is as well. This technique again uses the semilinearity of certain sets and a result that provides small preimages for morphisms from free commutative monoids to the integers.

The second technique is to show that \mathbb{B} is strongly λ -independent. Here, we use a construction that allows the postponement of increment operations and the early execution of decrement operations. This is then used to show that one can restrict oneself to computations in which a sequence of increments, followed by a sequence of decrements, will in the end change the counter only by a bounded amount.

The third technique is to show that if M is strongly λ -independent, where M is nontrivial, then $M * \mathbb{B}$ is as well. Here, the storage consists of a stack of elements of M . The construction works by encoding rational sets over $M * \mathbb{B}$ as elements on the stack. We have to use the semilinearity results again in order to be able to compute the set of all possible outputs when elements from two given rational sets cancel each other out (in the sense that push operations are followed by pop operations).

4 Semilinear Languages

This section contains semilinearity results that will be needed in later sections. The first lemma guarantees small preimages of morphisms from multisets to the integers. This will be used to bound the number of necessary operations on a blind counter in order to obtain a certain counter value.

Lemma 4.1. *Let $\varphi : X^\oplus \rightarrow \mathbb{Z}$ be a morphism. Then for any $n \in \mathbb{Z}$, the set $\varphi^{-1}(n)$ is semilinear. In particular, $\ker \varphi$ is finitely generated. Furthermore, there is a constant $k \in \mathbb{N}$ such that for any $\mu \in X^\oplus$, there is a $\nu \sqsubseteq \mu$ with $\mu \in \nu + \ker \varphi$ and $|\nu| \leq k \cdot |\varphi(\mu)|$.*

Proof. It is clearly possible to construct a context-free grammar for the language $L = \{w \in \{a, b\}^* \mid |w|_a - |w|_b = n\}$. Using an inverse homomorphism, one can then obtain a language K from L with $\Psi(K) = \varphi^{-1}(n)$. Thus, $\varphi^{-1}(n)$ is semilinear.

In order to prove the second claim, we present an algorithm to obtain ν from μ , from which it will be clear that the size of ν is linear in the absolute value of $\varphi(\mu)$. Without loss of generality, let $\varphi(\mu) > 0$. The algorithm operates in two phases.

In the first phase, we construct a $\nu \sqsubseteq \mu$ with $\varphi(\nu) \geq \varphi(\mu) - m$ such that $|\nu|$ is linear in $|\varphi(\mu)|$, where $m = \max\{|\varphi(x)| \mid x \in X\}$. In this phase, we start with $\nu = 0$ and successively add elements from μ to ν until $\varphi(\nu) \geq \varphi(\mu) - m$. As long as we still have $\varphi(\nu) < \varphi(\mu) - m$, it is guaranteed that we find an $x \in X$ such that $\nu + x \sqsubseteq \mu$ and $\varphi(x) > 0$. Thus, after at most $\varphi(\mu) - m$ steps, we have $\varphi(\nu) \geq \varphi(\mu) - m$ and $|\nu| \leq \varphi(\mu) - m$.

Since we stopped after we first had $\varphi(\nu) \geq \varphi(\mu) - m$, we also have $\varphi(\mu) - m \leq \varphi(\nu) \leq \varphi(\mu) + m$. In the second phase, we successively extend ν such that $\varphi(\nu)$ always stays within the interval $[\varphi(\mu) - m, \varphi(\mu) + m]$: If $\varphi(\nu) < \varphi(\mu)$, we can find an $x \in X$ with $\nu + x \sqsubseteq \mu$ and $\varphi(x) > 0$ and if $\varphi(\nu) > \varphi(\mu)$, we can find an $x \in X$ with $\nu + x \sqsubseteq \mu$ and $\varphi(x) < 0$. We do this nondeterministically and can therefore assume that no value $\varphi(\nu)$ occurs more than once: otherwise, we could have left out the summands between the two occurrences.

Hence, the values $\varphi(\nu)$ obtained in the course of the second phase are distinct numbers in $[\varphi(\mu) - m, \varphi(\mu) + m]$. Therefore, there is a computation in which after at most $2m + 1$ steps, we have $\varphi(\nu) = \varphi(\mu)$. Then clearly $|\nu| \leq \varphi(\mu) - m + 2m + 1$. \square

Another fact used in later sections is that languages in $\text{VA}(M)$ are semilinear if $M \in \mathcal{C}$. This will be used in various constructions, for instance when the effect of computations (that make use of M as storage) on the output in a commutative monoid is to be realized by a finite automaton. We prove this using a result of van Leeuwen [23]. He showed that semilinearity of all languages in a family is inherited by languages that are algebraic over this family. A language is called algebraic over a family of languages if it is generated by a grammar in which each production allows a non-terminal to be replaced by any word from a language in this family.

Definition 4.2. Let \mathcal{F} be a family of languages. An \mathcal{F} -grammar is a quadruple $G = (V, T, P, S)$ where V and $T \subseteq V$ are alphabets and $S \in V \setminus T$. P is a finite set of pairs

(A, M) with $A \in V \setminus T$ and $M \subseteq V^*$, $M \in \mathcal{F}$. In this context, a pair $(A, M) \in P$ will also be denoted by $A \rightarrow M$. We write $x \Rightarrow_G y$ if $x = uAv$ and $y = uwv$ for some $u, v, w \in V^*$ and $(A, M) \in P$ with $w \in M$. The language generated by G is $L(G) = \{w \in T^* \mid S \Rightarrow_G^* w\}$. A language L is called *algebraic over \mathcal{F}* if there is an \mathcal{F} -grammar G such that $L = L(G)$.

We will use the following result by van Leeuwen, which appeared in [23].

Theorem 4.3 (van Leeuwen). *Let \mathcal{F} be a family of semilinear languages. Then every language that is algebraic over \mathcal{F} is also semilinear.*

Note that in [19], a group G is called *SLI-group* if every language in $\text{VA}^\lambda(G)$ is semilinear (in different terms, however). Thus, the following recovers the result from [19], that the class of SLI-groups is closed under taking the free product.

Lemma 4.4. *Each $L \in \text{VA}^\lambda(M_0 * M_1)$ is algebraic over $\text{VA}^\lambda(M_0) \cup \text{VA}^\lambda(M_1)$.*

Proof. Let $L = L(A)$ for some valence automaton $A = (Q, X, M_0 * M_1, E, q_0, F)$. We assume that $E = E_0 \cup E_1$ with $E_i \subseteq Q \times X^* \times M_i \times Q$ and $F = \{q_f\}$. Let $V_i = \{K_{p,q}^{(i)} \mid p, q \in Q\}$ be a new alphabet for $i = 0, 1$. Furthermore, let $A_{p,q}^{(i)}$ be the automaton $(Q, X \cup V_{1-i}, M_i, E'_i, p, \{q\})$ with $E'_i = E_i \cup \{(r, K_{r,s}^{(1-i)}, 1, s) \mid r, s \in Q\}$. Finally, let A' be the automaton $(Q, X \cup V_0 \cup V_1, M_0 * M_1, E', q_0, F)$ with $E' = E \cup E'_0 \cup E'_1$. Then clearly $L(A) = L(A') \cap X^*$.

We will construct an \mathcal{F} -grammar $G = (V, X, P, S)$ such that $L(G) = L(A)$. This will be shown by proving that for any $w \in V^*$, we have $w \in L(A')$ iff $S \Rightarrow_G^* w$. Let $V = X \cup V_0 \cup V_1$ and P consist of the productions $K_{p,q}^{(i)} \rightarrow L(A_{p,q}^{(i)})$ for $p, q \in Q$ and $i \in \{0, 1\}$. Finally, let $S = K_{q_0, q_f}^{(0)}$. By induction on n , one can see that $S \Rightarrow_G^n w$ implies $w \in L(A')$. In particular, $w \in L(G)$ implies that w is accepted by A' without using the added edges in $E' \setminus E$. Thus, $L(G) \subseteq L(A)$.

Now suppose w is accepted by A' , which is witnessed by the edge sequence $s = u_0 v_1 u_1 \cdots v_n u_n$, where for some $i \in \{0, 1\}$, we have $u_0 \in E_i'^*$, $u_j \in E_i'^+$, $v_j \in E_{1-i}'^+$ for each $j \in \{1, \dots, n\}$.

We shall prove by induction on n that $S \Rightarrow_G^* w$. For $n = 0$, we have $w \in L(A_{q_0, q_f}^{(i)})$ and thus either $S = K_{q_0, q_f}^{(0)} \Rightarrow_G w$ or $S = K_{q_0, q_f}^{(0)} \Rightarrow_G K_{q_0, q_f}^{(1)} \Rightarrow_G w$. Therefore, let $n \geq 1$. Let $\varphi : E'^* \rightarrow X^*$, $\psi : E'^* \rightarrow M_0 * M_1$ be the morphisms with $\varphi((p, x, m, q)) = x$ and $\psi((p, x, m, q)) = m$. Then $\varphi(s) = w$ and $\psi(s) = 1$. Since $\psi(u_j) \in M_i$ and $\psi(v_j) \in M_{1-i}$, the definition of the free product yields that there is a j such that either $\psi(u_j) = 1$ or $\psi(v_j) = 1$, where $j > 0$ if $u_0 = \lambda$. Let $x \in E'^+$ be this word u_j or v_j . Note that since $\psi(x) = 1$, we have $\varphi(x) \in L(A_{p,q}^{(k)})$ for $k = i$ or $k = 1 - i$ (depending on whether $x = u_j$ or $x = v_j$) and where p and q are the initial and final state of the sequence x , respectively. We can obtain a sequence s' from s by replacing x with the edge $(p, K_{p,q}^{(k)}, 1, q)$. Clearly, s' is a valid sequence in A' and by induction, we have $S \Rightarrow_G^* \varphi(s')$. Since w can be obtained from $\varphi(s')$ by replacing $K_{p,q}^{(k)}$ with $\varphi(x) \in L(A_{p,q}^{(k)})$, we have $S \Rightarrow_G^* \varphi(s') \Rightarrow_G w$. \square

Combining the latter lemma with van Leeuwen's result and a standard argument for the preservation of semilinearity when building the direct product with \mathbb{Z} yields the following.

Lemma 4.5. *Let $M \in \mathcal{C}$. Then for any language $L \in \text{VA}^\lambda(M)$, the set $\Psi(L)$ is semilinear.*

Proof. Since $\text{VA}^\lambda(\mathbf{1})$ and $\text{VA}^\lambda(\mathbb{B})$ contain only context-free languages, the lemma holds for $M = \mathbf{1}$ and $M = \mathbb{B}$. Furthermore, by Lemma 4.4 and Theorem 4.3, we know that if $\text{VA}^\lambda(M)$ contains only semilinear languages then so does $\text{VA}^\lambda(M * \mathbb{B})$. Thus, it suffices to show that if every language in $\text{VA}^\lambda(M)$ is semilinear then so is every language in $\text{VA}^\lambda(M \times \mathbb{Z})$.

Let $L = L(A)$ for a valence automaton $A = (Q, X, M \times \mathbb{Z}, E, q_0, F)$ over $M \times \mathbb{Z}$. We assume that for each edge $(p, x, (m, z), q) \in E$, we have $x \in X \cup \{\lambda\}$. We construct an automaton A' as follows. Let $A' = (Q, X \times E, M, E', q_0, F)$, where $E' = \{(p, (x, e), m, q) \mid e = (p, x, (m, z), q) \in E\}$. Furthermore, let $\varphi : (X \times E)^\oplus \rightarrow \mathbb{Z}$ be the morphism with $\varphi((x, (p, y, (m, z), q))) = z$. Moreover, let $\pi : (X \times E)^\oplus \rightarrow X^\oplus$ be the projection map with $\pi((x, e)) = x$. Then by the definition of acceptance, we have $\Psi(L(A)) = \pi(\Psi(L(A'))) \cap \varphi^{-1}(0)$. Since $L(A')$ is semilinear by assumption, the class of semilinear subsets of a free commutative monoid is closed under intersection, and $\varphi^{-1}(0)$ is semilinear by Lemma 4.1, $L(A)$ is also semilinear. \square

5 Membership Problems

In this section, we study decidability and complexity of the membership problem for valence automata over $\mathbb{M}\Gamma$. Specifically, we show in this section that for certain graphs Γ , the class $\text{VA}^\lambda(\mathbb{M}\Gamma)$ contains undecidable languages (Lemma 5.5), while for every Γ , membership for languages in $\text{VA}(\mathbb{M}\Gamma)$ is (uniformly) decidable. We present two nondeterministic algorithms, one of them uses linear space and one runs in polynomial time (Lemma 5.4).

These results serve two purposes. First, for those graphs Γ that admit undecidable languages in $\text{VA}^\lambda(\mathbb{M}\Gamma)$, it follows that silent transitions are indispensable. Second, if we can show that silent transitions can be removed from valence automata over $\mathbb{M}\Gamma$, the algorithms also apply to languages in $\text{VA}^\lambda(\mathbb{M}\Gamma)$.

The algorithms in this section rely on the convergence property of certain reduction systems. For more information on reduction systems, see [1, 13]. A *reduction system* is a pair (S, \rightarrow) in which S is a set and \rightarrow is a binary relation on S . (S, \rightarrow) is said to be *noetherian* if there is no infinite sequence s_0, s_1, \dots with $s_i \rightarrow s_{i+1}$ for each $i \in \mathbb{N}$. We write $\xrightarrow{*}$ (\rightarrow^*) for the reflexive, transitive, symmetric (reflexive, transitive) closure of \rightarrow . (S, \rightarrow) has the *Church-Rosser property* if for any $s, t \in S$ with $s \xrightarrow{*} t$, there is a $u \in S$ with $s \xrightarrow{*} u$ and $t \xrightarrow{*} u$. We say that (S, \rightarrow) is *confluent*, if for any $s, t, u \in S$ with $s \xrightarrow{*} t$ and $s \xrightarrow{*} u$, there is a $v \in S$ with $t \xrightarrow{*} v$ and $u \xrightarrow{*} v$. A noetherian and confluent reduction system is called *convergent*. Furthermore, (S, \rightarrow) is called *locally confluent*, if for any $s, t, u \in S$ with $s \rightarrow t$ and $s \rightarrow u$, there is a $v \in S$ with $t \xrightarrow{*} v$ and $u \xrightarrow{*} v$. An

element $s \in S$ is *irreducible* if there is no $t \in S$ with $s \rightarrow t$. We say $t \in S$ is a *normal form* of $s \in S$ if $s \xrightarrow{*} t$ and t is irreducible. It is well-known that a reduction system is confluent if and only if it has the Church-Rosser property. Furthermore, a noetherian locally confluent reduction system is already confluent.

One of the steps in our algorithms will be to check, given a word $w \in X_{\Gamma}^*$, whether $w \equiv_{\Gamma} \lambda$. Unfortunately, turning the Thue system T_{Γ} into a reduction system on words will not yield a convergent reduction system as the length-preserving rules allow for infinite reduction sequences. Therefore, we will use reduction systems on traces instead. For more information on traces, see [4].

Let X be an alphabet. An irreflexive symmetric relation $I \subseteq X \times X$ is called an *independence relation*. To each such relation, the corresponding Thue system $T_I = (X, R_I)$ is given as $R_I = \{(ab, ba) \mid (a, b) \in I\}$. If \equiv_I denotes the congruence generated by T_I , then the monoid $\mathbb{T}(X, I) = X^* / \equiv_I$ is called *trace monoid*, its elements *traces*. The equivalence class of $u \in X^*$ is denoted as $[u]_I$ and since the words in an equivalence class all have the same length, $|[u]_I| = |u|$ is well-defined.

In order to efficiently compute using traces, we represent them using dependence graphs. Let X be an alphabet and $I \subseteq X \times X$ an independence relation. To each word $w \in X^*$ we assign a loop-free directed acyclic vertex-labeled graph, its *dependence graph* $\text{dep}(w)$. If $w = x_1 \cdots x_n$, $x_i \in X$, $1 \leq i \leq n$, then $\text{dep}(w) = (V, E, \ell)$, in which $E \subseteq V \times V$, has vertex set $V = \{1, \dots, n\}$ and $(i, j) \in E$ if and only if $i < j$ and $(x_i, x_j) \notin I$. Furthermore, each vertex i is labeled with $\ell(i) = x_i \in X$. It is well-known that for words $u, v \in X^*$, we have $u \equiv_I v$ if and only if $\text{dep}(u)$ and $\text{dep}(v)$ are isomorphic. Thus, we will also write $\text{dep}(s)$ for $\text{dep}(u)$ if $s = [u]_I$.

Each (undirected, potentially looped) graph $\Gamma = (V, E)$ gives rise to an independence relation on X_{Γ} , namely

$$I = \{(x, y) \mid x \in \{a_v, \bar{a}_v\}, y \in \{a_w, \bar{a}_w\}, x \neq y, \{v, w\} \in E\}. \quad (1)$$

If I is given by Γ in this way, we also write $\equiv_{\Gamma|\mathbb{T}}$ for \equiv_I and $[u]_{\Gamma|\mathbb{T}}$ instead of $[u]_I$.

In the following, let I be given by $\Gamma = (V, E)$ as in (1). We will now define a reduction relation \rightarrow on $\mathbb{T}(X_{\Gamma}, I)$ such that for $u, v \in X_{\Gamma}^*$

$$[u]_{\Gamma} = [v]_{\Gamma} \quad \text{if and only if} \quad [u]_{\Gamma|\mathbb{T}} \xleftarrow{*} [v]_{\Gamma|\mathbb{T}}. \quad (2)$$

For $s, t \in \mathbb{T}(X_{\Gamma}, I)$, let $s \rightarrow t$ if there are $u_1, u_2 \in X_{\Gamma}^*$ and $v \in V$ such that $s = [u_1 a_v \bar{a}_v u_2]_{\Gamma|\mathbb{T}}$ and $t = [u_1 u_2]_{\Gamma|\mathbb{T}}$. This definition immediately yields (2). Since our algorithms will represent traces as dependence graphs, we have to restate this relation in terms of the latter. It is not hard to see that for $s, t \in \mathbb{T}(X_{\Gamma}, I)$, $s \rightarrow t$ if and only if there are vertices x, y in $\text{dep}(s)$, labeled a_v and \bar{a}_v , respectively, such that

1. there is no path from y to x and
2. there is no vertex lying on a path from x to y

and $\text{dep}(t)$ is obtained from $\text{dep}(s)$ by deleting x and y . We will refer to conditions 1 and 2 as the *subtrace conditions*.

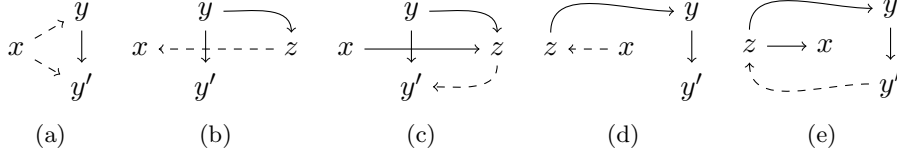


Figure 1: Possible fragments of the dependence graph of s .

Lemma 5.1. *The reduction system $(\mathbb{T}(X_\Gamma, I), \rightarrow)$ is convergent.*

Proof. Since the system is clearly noetherian, it remains to be shown that $(\mathbb{T}(X_\Gamma, I), \rightarrow)$ is locally confluent. Hence, let x, y, x', y' be vertices in $\text{dep}(s)$ labeled $a_v, \bar{a}_v, a_w, \bar{a}_w$, respectively, satisfying the subtrace conditions such that $\text{dep}(t)$ is obtained by deleting x, y and $\text{dep}(t')$ is obtained by deleting x', y' . If $\{x, y\} = \{x', y'\}$, we are done. Furthermore, if $\{x, y\} \cap \{x', y'\} = \emptyset$, deleting x, y from $\text{dep}(t')$ (or x', y' from $\text{dep}(t)$) yields a $u \in \mathbb{T}(X_\Gamma, I)$ with $t \rightarrow u$ and $t' \rightarrow u$. Therefore, we assume $x = x'$ and $y \neq y'$ (the case $x \neq x', y = y'$ can be done analogously). This means in particular that $v = w$. Since $(\bar{a}_v, \bar{a}_v) \notin I$, we can also assume that there is an edge from y to y' .

If $(a_v, \bar{a}_v) \notin I$, there are edges (x, y) and (x, y') in $\text{dep}(s)$ and y violates the second subtrace condition of x, y' (see Figure 1a). Hence, we have $(a_v, \bar{a}_v) \in I$. We claim that flipping y and y' constitutes an automorphism of $\text{dep}(s)$, meaning $\text{dep}(t)$ and $\text{dep}(t')$ are isomorphic and thus $t = t'$. The former amounts to showing that each vertex z in $\text{dep}(s)$ has an edge from (to) y iff z has one from (to) y' .

If there is an edge from y to z , then by the definition of I , we also have an edge between x and z . Obeying the first subtrace condition, it has to be directed from x to z : Otherwise, there would be a path from y to x (see Figure 1b). Since y and y' share the same label, we also have an edge between y' and z . If this were an edge from z to y' , z would lie on a path from $x = x'$ to y' (see Figure 1c), violating the second subtrace condition. Hence, there is an edge from y' to z .

If there is an edge from z to y , then by the definition of I , we also have an edge between x and z . By the second subtrace condition, it has to be directed from z to x : Otherwise, z would lie on a path from x to y (see Figure 1d). Since y and y' share the same label, we also have an edge between y' and z . If this were directed from y' to z , then there would be a path from y' to $x = x'$ (see Figure 1e), violating the first subtrace condition. Hence, there is an edge from z to y' .

If there is no edge between y and z , there is also no edge between y' and z , since y and y' have the same label. \square

By (2) and since $(\mathbb{T}(X_\Gamma, I), \rightarrow)$ is convergent, we have

$$[w]_\Gamma = [\lambda]_\Gamma \text{ if and only if } [w]_{\Gamma|\mathbb{T}} \xrightarrow{*} [\lambda]_{\Gamma|\mathbb{T}}. \quad (3)$$

This equivalence is the basis of our algorithms to check for the former condition.

Lemma 5.2. *There is a deterministic polynomial-time algorithm that, given a word $w \in X_\Gamma^*$, determines whether $[w]_\Gamma = [\lambda]_\Gamma$.*

Proof. By (3), the condition $[w]_\Gamma = [\lambda]_\Gamma$ is equivalent to $[\lambda]_{\Gamma|\mathbb{T}}$ being the normal form of $[w]_{\Gamma|\mathbb{T}}$. Therefore, our algorithm computes the normal form of $[w]_{\Gamma|\mathbb{T}}$. It does so by computing the dependence graph of w and successively deleting pairs of nodes that satisfy the subtrace conditions. Finding such a pair can be done in polynomial time and since at most $|w|/2$ deletions are possible, the normal form is obtained after polynomial time. In the end, the algorithm checks whether the calculated dependence graph representing the normal form is empty. \square

Lemma 5.3. *There is a nondeterministic linear-space algorithm that, given a word $w \in X_\Gamma^*$, determines whether $[w]_\Gamma = [\lambda]_\Gamma$.*

Proof. Let $\Gamma = (V, E)$. By (3), we have $[w]_\Gamma = [\lambda]_\Gamma$ if and only if w can be reduced to the empty word by commuting a_v and a_w for $v, w \in E$, commuting a_v and \bar{a}_v for $\{v\} \in E$ and deleting $a_v \bar{a}_v$ for $v \in V$. Therefore, we can clearly construct a monotone grammar for the set of all $w \in X_\Gamma^*$ that have $[w]_\Gamma = [\lambda]_\Gamma$. \square

Lemma 5.4. *For each $L \in \text{VA}(\mathbb{M}\Gamma)$, the membership problem can be decided by*

1. *a nondeterministic polynomial-time algorithm as well as*
2. *a nondeterministic linear-space algorithm.*

In particular, the languages in $\text{VA}(\mathbb{M}\Gamma)$ are context-sensitive.

Proof. In order to decide the membership problem for a word w for a language in $\text{VA}(\mathbb{M}\Gamma)$, we can guess a run reading w . Since there are no λ -transitions in the automaton, such a run has length linear in $|w|$. For this run, we have to check whether the product of the monoid elements on the edges is the identity element of $\mathbb{M}\Gamma$. By lemmas 5.2 and 5.3, this can be done in polynomial time or using linear space. Hence, the lemma follows. \square

Lemma 5.5. *Let Γ be a graph whose underlying loop-free graph is a path on four vertices. Then $\text{VA}^\lambda(\mathbb{M}\Gamma)$ contains an undecidable language.*

Proof. Let $\Gamma = (V, E)$ and $\hat{\Gamma}$ be the graph obtained from Γ by adding a loop to every unlooped vertex. For notational reasons, we assume that the vertex set of $\hat{\Gamma}$ is $\hat{V} = \{\hat{v} \mid v \in V\}$. Lohrey and Steinberg [19] show that there are rational sets $\hat{R}, \hat{S} \subseteq \mathbb{M}\hat{\Gamma}$ over positive generators such that given a word $w \in \mathbb{M}\hat{\Gamma}$ over positive generators, it is undecidable whether $1 \in w\hat{R}\hat{S}^{-1}$. Note that the morphism $\varphi : \mathbb{M}\Gamma \rightarrow \mathbb{M}\hat{\Gamma}$ with $\varphi(a_v) = a_{\hat{v}}$ and $\varphi(\bar{a}_v) = \bar{a}_{\hat{v}}$ induces an isomorphism between the submonoids generated by positive generators and between the submonoids generated by the negative generators. Thus, we find rational sets $R, S \subseteq \mathbb{M}\Gamma$ over positive generators with $\varphi(R) = \hat{R}$ and $\varphi(S) = \hat{S}$.

If w is a word over positive generators in $\mathbb{M}\Gamma$, $w = a_1 \cdots a_n$, then we let $\bar{w} = \bar{a}_n \cdots \bar{a}_1$. This is well-defined, for if $a_1 \cdots a_n = b_1 \cdots b_m$, for positive generators $a_1, \dots, a_n, b_1, \dots, b_m$ then $\varphi(a_1 \cdots a_n) = \varphi(b_1 \cdots b_m)$ and thus $\varphi(\bar{a}_n \cdots \bar{a}_1) = \varphi(a_1 \cdots a_n)^{-1} = \varphi(b_1 \cdots b_m)^{-1} = \varphi(\bar{b}_m \cdots \bar{b}_1)$ and therefore $\bar{a}_n \cdots \bar{a}_1 = \bar{b}_m \cdots \bar{b}_1$. Note that $w\bar{w} = 1$ for every word w over

positive generators. With this definition, the set $\bar{S} = \{\bar{s} \mid s \in S\}$ is also rational. We claim that for a word $w \in \mathbb{M}\Gamma$ over positive generators, $1 \in wR\bar{S}$ if and only if $1 \in \varphi(w)\mathring{R}\mathring{S}^{-1}$.

If $1 \in \varphi(w)\mathring{R}\mathring{S}^{-1}$, there are $\mathring{r} \in \mathring{R}$, $\mathring{s} \in \mathring{S}$ with $1 = \varphi(w)\mathring{r}\mathring{s}^{-1}$ and thus $\mathring{s} = \varphi(w)\mathring{r}$. Thus, we can find $s \in S$ and $r \in R$ with $\varphi(s) = \varphi(w)\varphi(r)$. The injectivity of φ on words over positive generators yields $s = wr$ and thus $1 = wr\bar{s}$. Hence $1 \in wR\bar{S}$.

If $1 \in wR\bar{S}$, we have $1 = wr\bar{s}$ for some $r \in R$ and $s \in S$. This implies $1 = \varphi(w)\varphi(r)\varphi(s)^{-1}$ and since $\varphi(r) \in \mathring{R}$ and $\varphi(s)^{-1} \in \mathring{S}^{-1}$, we have $1 \in \varphi(w)\mathring{R}\mathring{S}^{-1}$.

Thus, given a word $w \in \mathbb{M}\Gamma$ over positive generators, it is undecidable whether $1 \in wR\bar{S}$. Now, we construct a valence automaton over $\mathbb{M}\Gamma$ that reads a representative of a word w and then nondeterministically multiplies an element from R and then an element from \bar{S} . It accepts if and only if $1 \in wR\bar{S}$. Therefore, the automaton accepts an undecidable language. \square

6 Rational Sets

Here, we present a normal form result for rational subsets of monoids in \mathcal{C} . The first lemma is a simple observation for which we will not provide a proof.

Lemma 6.1. *Let $s \in M * \mathbb{B}$ and $s = m_1 \bar{a} \cdots m_k \bar{a} m_1' \cdots m_\ell'$. Then we have*

- $s \in J(M * \mathbb{B})$ if and only if $m_i \in L(M)$, $m \in J(M)$, $m_j' \in R(M)$ for all $1 \leq i \leq k$, $1 \leq j \leq \ell$.
- $s \in L(M * \mathbb{B})$ if and only if $\ell = 0$ and $m_i, m \in L(M)$ for $1 \leq i \leq k$.
- $s \in R(M * \mathbb{B})$ if and only if $k = 0$ and $m_i', m \in R(M)$ for $1 \leq i \leq \ell$.

The following lemma states the normal form result. Note that in a valence automaton over M , we can remove all edges labeled with elements outside of $J(M)$. This is due to the fact that they cannot be part of a valid computation. In a valence transducer over M with output in C , the edges carry elements from $X^* \times M \times C$, in which M is used as a storage. Therefore, of a rational set $S \subseteq M \times C$, we will only be interested in the part $S \cap (J(M) \times C)$.

If $A = (Q, M, E, q_0, F)$ is an automaton over M and $B = (Q', M, E', q_0', \{q_f'\})$ is an automaton over M with only one final state and $Q \cap Q' = \emptyset$, then the automaton obtained by *gluing in* B between $p, q \in Q$ is $C = (Q \cup Q', M, E'', q_0, F)$, where

$$E'' = E \cup E' \cup \{(p, 1, q_0'), (q, 1, q_f')\}.$$

Lemma 6.2. *Let $M \in \mathcal{C}$ and C be a commutative monoid and $S \subseteq M \times C$ a rational set. Then we have $S \cap (J(M) \times C) = \bigcup_{i=1}^n L_i U_i R_i$, in which*

- (i) $L_i \in \text{RAT}(L(M) \times C)$,
- (ii) $U_i \in \text{RAT}(H(M) \times C)$, and

(iii) $R_i \in \text{RAT}(\text{R}(M) \times C)$

for $1 \leq i \leq n$. Moreover,

$$S \cap (\text{L}(M) \times C) = \bigcup_{1 \leq i \leq n, 1 \in R_i} L_i U_i, \quad (4)$$

$$S \cap (\text{R}(M) \times C) = \bigcup_{1 \leq i \leq n, 1 \in L_i} U_i R_i. \quad (5)$$

Proof. We proceed by induction. Since the lemma clearly holds for $M = \mathbf{1}$, we show that if it holds for M , it is also true for $M \times \mathbb{Z}$ and $M * \mathbb{B}$. Therefore, suppose the lemma holds for M .

Let $S \subseteq (M \times \mathbb{Z}) \times C$ be a rational subset. We regard S as a subset of $M \times (\mathbb{Z} \times C)$. By induction, we have $S \cap (\text{J}(M) \times (\mathbb{Z} \times C)) = \bigcup_{i=1}^n L_i U_i R_i$ with the properties above. Since $\text{X}(M \times \mathbb{Z}) = \text{X}(M) \times \mathbb{Z}$ where X is any of the operators $\text{J}, \text{H}, \text{L}, \text{R}$, the sets L_i, U_i, R_i can serve as the desired decomposition for $(M \times \mathbb{Z}) \times C$.

Let $S \subseteq (M * \mathbb{B}) \times C$ be rational. Then there is an alphabet X , a rational language $L \subseteq X^*$, and a morphism $\varphi : X^* \rightarrow (M * \mathbb{B}) \times C$ with $\varphi(L) = S$. Without loss of generality, we assume that $X = \{x, \bar{x}\} \cup Y \cup Z$ with $\varphi(x) = a$, $\varphi(\bar{x}) = \bar{a}$, $\varphi(Y) \subseteq M$, $\varphi(Z) \subseteq C$, where a and \bar{a} are the two generators of \mathbb{B} . Let A be an automaton accepting L such that every edge carries exactly one letter.

As a first step, we will construct an automaton A' that also has $\varphi(L(A')) = S$ but which has for every element of $S \cap (\text{J}(M * \mathbb{B}) \times C)$ a representative in $X^* \setminus X^* x X^* \bar{x} X^*$.

Let $A = (Q, X, E, q_0, F)$. For $p, q \in Q$, the language

$$K_{p,q} = \{\pi_Z(w) \mid w \in L_{p,q}(A), \varphi(w) \in \{1\} \times C\}$$

is clearly contained in $\text{VA}^\lambda(M * \mathbb{B})$ and is therefore semilinear by Lemma 4.5. Thus, we can find a finite automaton $A'_{p,q}$ such that $\Psi(L(A'_{p,q})) = \Psi(K_{p,q})$. Since C is commutative and $\varphi(Z) \subseteq C$, this also means $\varphi(L(A'_{p,q})) = \varphi(K_{p,q})$. The automaton A' is now obtained from A by gluing $A'_{p,q}$ into A between p and q , for each $p, q \in Q$. Since in A' for each path from the initial to the final state, we can find another path that encodes the same element of $(M * \mathbb{B}) \times C$ and is present in A , we have $\varphi(L(A')) = \varphi(L(A)) = S$. However, the glued in automata allow us to encode elements of $S \cap (\text{J}(M * \mathbb{B}) \times C)$ by words of a certain form. Specifically, we claim that

$$S \cap (\text{J}(M * \mathbb{B}) \times C) \subseteq \varphi(L(A') \setminus X^* x X^* \bar{x} X^*) \subseteq S. \quad (6)$$

Let $s \in S \cap (\text{J}(M * \mathbb{B}) \times C)$ and $w \in L(A')$ be chosen such that $\varphi(w) = s$ and $|w|_{X \setminus Z}$ is minimal. Toward a contradiction, suppose $w \in X^* x X^* \bar{x} X^*$. Then $w = f x g \bar{x} h$ with $f, h \in X^*$, $g \in (Y \cup Z)^*$. Since $a m \bar{a} \notin \text{J}(M * \mathbb{B})$ for any $m \in M \setminus \{1\}$, our assumption $s \in \text{J}(M * \mathbb{B}) \times C$ implies $\varphi(g) \in \{1\} \times C$ and thus $\varphi(x g \bar{x}) \in \{1\} \times C$. By the construction of A' , however, this means that there is a word $v \in Z^*$ such that $f v h \in L(A')$ and $\varphi(f v h) = \varphi(w)$. Since $|f v h|_{X \setminus Z} < |w|_{X \setminus Z}$, this contradicts the choice of w , proving (6).

Let $A'' = (Q'', X, E'', q_0'', \{q_f''\})$ be a spelling finite automaton accepting $L(A') \setminus X^* x X^* \bar{x} X^*$ with input alphabet X . Furthermore, for each $p, q \in Q''$, let $B_p^{(-)}$, $B_{p,q}^{(0)}$, $B_q^{(+)}$ be spelling automata satisfying

$$\begin{aligned} L(B_p^{(-)}) &= L_{q_0'', p}(A'') \cap ((Y \cup Z)^* \bar{x})^* \\ L(B_{p,q}^{(0)}) &= L_{p,q}(A'') \cap (Y \cup Z)^* \\ L(B_q^{(+)}) &= L_{q, q_f''}(A'') \cap (x(Y \cup Z)^*)^* \end{aligned}$$

Then we have

$$L(A'') = \bigcup_{p,q \in Q''} L(B_p^{(-)}) L(B_{p,q}^{(0)}) L(B_q^{(+)}). \quad (7)$$

Since $S \cap (J(M * \mathbb{B}) \times C) \subseteq \varphi(L(A'')) \subseteq S$, we will now modify A'' so as to accept only words whose image lies in $S \cap (J(M * \mathbb{B}) \times C)$. This will be achieved by exploiting the induction hypothesis for $M \times C$. From $B_p^{(-)}$, we obtain $\tilde{B}_p^{(-)}$ by removing all edges with letters in $Y \cup Z$ and then between any two states r, s gluing in an automaton that accepts a rational language with image

$$\varphi(L_{r,s}(B_p^{(-)}) \cap (Y \cup Z)^*) \cap (L(M) \times C),$$

which exists by induction. The automaton $\tilde{B}_q^{(+)}$ is obtained from $B_q^{(+)}$ in an analogous way: we remove edges with letters in $Y \cup Z$ and between any two states r, s , glue in an automaton that accepts a rational language with image

$$\varphi(L_{r,s}(B_q^{(+)}) \cap (Y \cup Z)^*) \cap (R(M) \times C).$$

By the induction hypothesis, we can assume that the input alphabet of $\tilde{B}_p^{(-)}$ and $\tilde{B}_q^{(+)}$ is $\{\bar{x}\} \cup Y^{(-)} \cup Z$ and $\{x\} \cup Y^{(+)} \cup Z$, respectively, where $\varphi(Y^{(-)}) \subseteq L(M) \times C$ and $\varphi(Y^{(+)}) \subseteq R(M) \times C$. Let $K_p^{(-)} = \varphi(L(\tilde{B}_p^{(-)}))$, $K_q^{(+)} = \varphi(L(\tilde{B}_q^{(+)}))$. Then $K_p^{(-)} \in \text{RAT}(L(M * \mathbb{B}) \times C)$, $K_q^{(+)} \in \text{RAT}(R(M * \mathbb{B}) \times C)$ and

$$\begin{aligned} K_p^{(-)} &= \varphi(L(B_p^{(-)})) \cap (L(M * \mathbb{B}) \times C), \\ K_q^{(+)} &= \varphi(L(B_q^{(+)})) \cap (R(M * \mathbb{B}) \times C). \end{aligned}$$

Finally, the induction hypothesis provides for each $p, q \in Q''$ rational sets $L_{p,q,i} \in \text{RAT}(L(M) \times C)$, $U_{p,q,i} \in \text{RAT}(H(M) \times C)$, $R_{p,q,i} \in \text{RAT}(R(M) \times C)$ such that

$$\varphi(L(B_{p,q}^{(0)})) \cap (J(M) \times C) = \bigcup_{i=1}^{n(p,q)} L_{p,q,i} U_{p,q,i} R_{p,q,i}.$$

We claim that for $\tilde{S} = S \cap (J(M * \mathbb{B}) \times C)$, we have

$$\tilde{S} = \bigcup_{\substack{p,q \in Q'' \\ 1 \leq i \leq n(p,q)}} \left[K_p^{(-)} L_{p,q,i} \right] U_{p,q,i} \left[R_{p,q,i} K_q^{(+)} \right]. \quad (8)$$

Thus, let $s \in \tilde{S}$. By (6), the definition of A'' , and (7), we can write $s = \varphi(w)$ with

$$w = u_1 \bar{x} \cdots u_k \bar{x} v x w_1 \cdots x w_\ell$$

and $u_i, v, w_j \in (Y \cup Z)^*$ for $1 \leq i \leq k$, $1 \leq j \leq \ell$ and $u_1 \bar{x} \cdots u_k \bar{x} \in L(B_p^{(-)})$, $v \in L(B_{p,q}^{(0)})$, $x w_1 \cdots x w_\ell \in L(B_q^{(+)})$ for some $p, q \in Q''$. By Lemma 6.1, the fact that $s \in J(M * \mathbb{B}) \times C$ implies that $\varphi(u_i) \in L(M) \times C$, $\varphi(v) \in J(M) \times C$, and $\varphi(w_j) \in R(M) \times C$ for $1 \leq i \leq k$, $1 \leq j \leq \ell$. In particular, there are words $\tilde{u}_i \in (Y^{(-)} \cup Z)^*$, $\tilde{w}_j \in (Y^{(+)} \cup Z)^*$ such that $\varphi(\tilde{u}_i) = \varphi(u_i)$, $\varphi(\tilde{w}_j) = \varphi(w_j)$ and

$$\tilde{u}_1 \bar{x} \cdots \tilde{u}_k \bar{x} \in L(\tilde{B}_p^{(-)}), \quad x \tilde{w}_1 \cdots x \tilde{w}_\ell \in L(\tilde{B}_q^{(+)})$$

and $\varphi(v) \in L_{p,q,i} U_{p,q,i} R_{p,q,i}$ for some $1 \leq i \leq n(p, q)$, proving “ \subseteq ” of (8).

For the inclusion “ \supseteq ”, note that by (6) and the definition of the sets on the right, the right side is contained in S . Moreover, by Lemma 6.1, the right side is also contained in $J(M * \mathbb{B}) \times C$. This proves (8).

We will show that (8) is the desired decomposition of \tilde{S} . We have already established that

- (i) $K_p^{(-)} L_{p,q,i} \in \text{RAT}(L(M * \mathbb{B}) \times C)$,
- (ii) $U_{p,q,i} \in \text{RAT}(H(M) \times C) \subseteq \text{RAT}(H(M * \mathbb{B}) \times C)$, and
- (iii) $R_{p,q,i} K_q^{(+)} \in \text{RAT}(R(M * \mathbb{B}) \times C)$.

Therefore, it remains to be shown that (4) and (5) are satisfied. We only prove (4), the proof for (5) can be done analogously. The inclusion “ \supseteq ” is immediately clear in each case. Thus, suppose $s \in S \cap (L(M * \mathbb{B}) \times C)$. Then there is a word $w \in L(A'')$ with $s = \varphi(w)$ and

$$w = u_1 \bar{x} \cdots u_k \bar{x} v x w_1 \cdots x w_\ell$$

and $u_i, v, w_j \in (Y \cup Z)^*$ for $1 \leq i \leq k$, $1 \leq j \leq \ell$ and $u_1 \bar{x} \cdots u_k \bar{x} \in L(B_p^{(-)})$, $v \in L(B_{p,q}^{(0)})$, $x w_1 \cdots x w_\ell \in L(B_q^{(+)})$ for some $p, q \in Q''$. By Lemma 6.1, the fact that $s \in R(M * \mathbb{B}) \times C$ implies $\varphi(u_i), \varphi(v) \in L(M) \times C$ for $1 \leq i \leq k$ and $\ell = 0$. In particular, there are words $\tilde{u}_i \in (Y^{(-)} \cup Z)^*$ such that $\varphi(\tilde{u}_i) = \varphi(u_i)$ and

$$\tilde{u}_1 \bar{x} \cdots \tilde{u}_k \bar{x} \in L(\tilde{B}_p^{(-)}), \quad \lambda \in L(\tilde{B}_q^{(+)})$$

and $\varphi(v) \in L_{p,q,i} U_{p,q,i}$ and $1 \in R_{p,q,i}$ for some $1 \leq i \leq n(p, q)$ by induction. Therefore, $1 \in R_{p,q,i} K_q^{(+)}$ and $s \in K_p^{(-)} L_{p,q,i} U_{p,q,i}$. This proves the remaining inclusion “ \subseteq ” of (4). \square

If we regroup the factors in (8), we obtain

$$\tilde{S} = \bigcup_{\substack{p,q \in Q'' \\ 1 \leq i \leq n(p,q)}} K_p^{(-)} [L_{p,q,i} U_{p,q,i} R_{p,q,i}] K_q^{(+)},$$

which implies the following corollary. Note, however, that it can also be deduced from Lemma 6.2.

Corollary 6.3. *Let $M \in \mathcal{C}$ and S be a rational subset of $(M * \mathbb{B}) \times C$. Then there is an alphabet $X = \{x, \bar{x}\} \cup Y \cup Z$, a morphism $\varphi : X^* \rightarrow (M * \mathbb{B}) \times C$ with $\varphi(x) = a$, $\varphi(\bar{x}) = \bar{a}$, $\varphi(Y) \subseteq M$, $\varphi(Z) \subseteq C$, a number $n \in \mathbb{N}$, and rational languages*

$$L_i \subseteq ((Y \cup Z)^* \bar{x})^*, \quad J_i \subseteq (Y \cup Z)^*, \quad R_i \subseteq (x(Y \cup Z)^*)^*$$

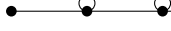
such that

$$S \cap (J(M * \mathbb{B}) \times C) = \bigcup_{i=1}^n \varphi(L_i) \varphi(J_i) \varphi(R_i).$$

7 Silent Transitions

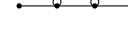
In this section, we use the facts established in earlier sections to prove the main results.

Lemma 7.1. *Let Γ be a graph such that*

- *between any two looped vertices, there is an edge, and*
- *between any two unlooped vertices, there is no edge, and*
- *Γ does not contain  as an induced subgraph.*

Then $\mathbb{M}\Gamma$ is in \mathcal{C} .

Proof. We proceed by induction and thus assume that $\mathbb{M}(\Gamma \setminus \{x\}) \in \mathcal{C}$ for any vertex x . Let $\Gamma = (V, E)$ and write $V = L \cup U$, where L is the set of looped vertices and U is the set of unlooped vertices. For every $x \in L$, let $\nu(x) = N(x) \cap U$, i.e. the set of unlooped neighbors of x . We write $x \leq y$ for $x, y \in L$ if $\nu(x) \subseteq \nu(y)$. Clearly, \leq is a reflexive, transitive order on L .

If there were $x, y \in L$ such that $\nu(x)$ and $\nu(y)$ are incomparable, there would be vertices $u, v \in U$ with $u \in \nu(x) \setminus \nu(y)$ and $v \in \nu(y) \setminus \nu(x)$. Thus, the vertices u, x, y, v induce the subgraph , contradicting the hypothesis. Hence, \leq is a total order and has a greatest element $g \in L$.

- If $\nu(g) = U$, then g is adjacent to every vertex in Γ and thus $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus \{g\}) \times \mathbb{Z}$.
- If $\nu(g) \subsetneq U$, then there is an isolated vertex $u \in U \setminus \nu(g)$. Hence, we have $\mathbb{M}\Gamma \cong \mathbb{M}(\Gamma \setminus \{u\}) * \mathbb{B}$.

□

We will prove Theorem 3.1 by showing that $\text{VA}(M) = \text{VA}^\lambda(M)$ for every $M \in \mathcal{C}$. This will be done using an induction with respect to the definition of \mathcal{C} . In order for this induction to work, we need to strengthen the induction hypothesis. The latter will state that for any $M \in \mathcal{C}$ and any commutative monoid C , we can transform a valence transducer over M with output in C into another one that has no λ -transitions but is allowed to output a semilinear set of elements in each step. Formally, we will show that each $M \in \mathcal{C}$ is *strongly λ -independent*.

Definition 7.2. Let C be a commutative monoid and $T \subseteq X^* \times \text{SL}(C)$ be a transduction. Then $\Phi(T) \subseteq X^* \times C$ is defined as

$$\Phi(T) = \{(w, c) \in X^* \times C \mid \exists (w, S) \in T : c \in S\}.$$

For a class \mathcal{F} of transductions, $\Phi(\mathcal{F})$ is the class of all $\Phi(T)$ with $T \in \mathcal{F}$.

A monoid M is called *strongly λ -independent* if for any commutative monoid C , we have $\text{VT}^\lambda(M, C) = \Phi(\text{VT}(M, \text{SL}(C)))$.

Note that the inclusion $\Phi(\text{VT}(M, \text{SL}(C))) \subseteq \text{VT}^\lambda(M, C)$ holds for any M and C . Here, in order to have equality, it is necessary to grant the λ -free transducer the output of semilinear sets, since valence transducers without λ -transitions and with output in C can only output finitely many elements per input word. With λ -transitions, however, a valence transducer can output an infinite set for one input word.

By choosing the trivial monoid for C , we can see that for every strongly λ -independent monoid M , we have $\text{VA}(M) = \text{VA}^\lambda(M)$. Indeed, given a valence automaton A over M , add an output of 1 to each edge and transform the resulting valence transducer into a λ -free one with output in $\text{SL}(\mathbf{1})$. The latter can then clearly be turned into a valence automaton for the language accepted by A . Thus, we have the following lemma.

Lemma 7.3. *If M is strongly λ -independent, then $\text{VA}(M) = \text{VA}^\lambda(M)$.*

Definition 7.4. A *rationally labeled valence transducer* over M with output in C is an automaton over $X^* \times \text{RAT}(M \times C)$. For $A = (Q, X^* \times \text{RAT}(M \times C), E, q_0, F)$, we also write $A = (Q, X, M, C, E, q_0, F)$. The *transduction performed by A* is

$$T(A) = \{(w, c) \in X^* \times C \mid \exists q \in F : (q_0, (\lambda, \{1\})) \Rightarrow_A^* (q, (w, S)), (1, c) \in S\}.$$

A is called *spelling* if $E \subseteq Q \times X \times \text{RAT}(M \times C) \times Q$, i.e., if it reads exactly one letter in each transition.

The definition of $T(A)$ for rationally labeled valence transducers A means that A behaves as if instead of an edge $(p, (w, S), q)$, $S \in \text{RAT}(M \times C)$, it had an edge (p, w, m, c, q) for each $(m, c) \in S$. Therefore, in slight abuse of terminology, we will also say that

$$q_0 \xrightarrow{(x_1, m_1, c_1)} q_1 \rightarrow \cdots \rightarrow q_{n-1} \xrightarrow{(x_n, m_n, c_n)} q_n$$

is a computation in A when there are edges $(q_{i-1}, (x_i, S_i), q_i) \in E$ such that $(m_i, c_i) \in S_i$ for $1 \leq i \leq n$.

Lemma 7.5. *For each valence transducer A over M with output in C , there is a spelling rationally labeled valence transducer A' with $T(A') = T(A)$.*

Proof. Let $A = (Q, X, M, C, E, q_0, F)$. We obtain the λ -free rationally labeled valence transducer $A' = (Q, X, M, C, E', q_0, F)$ as follows. We introduce one edge $(p, (x, S), q)$ for every triple $(p, x, q) \in Q \times X \times Q$ such that $S \subseteq M \times C$ is the rational set of elements spelled by paths in A that start in p , go along a number of λ -edges, then pass through an edge labeled x and then again go along a number of λ -edges and stop in q . Then clearly $T(A') = T(A)$. \square

Lemma 7.6. \mathbb{B} is strongly λ -independent.

Proof. Let $T \in \text{VT}^\lambda(\mathbb{B}, C)$. By Lemma 7.5, we can assume that $T = T(A)$ for a rationally labeled valence transducer $A = (Q, X, \mathbb{B}, C, E, q_0, F)$ over \mathbb{B} with output in C .

By Lemma 6.2, we can assume that every edge in A has the form (p, x, LR, q) , with $L \in \text{RAT}(\{\bar{a}\}^\oplus \times C)$ and $R \in \text{RAT}(\{a\}^\oplus \times C)$. Furthermore, we can assume that edges starting in the initial state q_0 are of the form (q_0, x, R, p) and, analogously, edges ending in a final state $q \in F$ are of the form (p, x, L, q) , $p \in Q$, $L \in \text{RAT}(\{\bar{a}\}^\oplus \times C)$ and $R \in \text{RAT}(\{a\}^\oplus \times C)$. Thus, we can construct an equivalent transducer $A' = (Q', X, \mathbb{B}, C, E', q_0, F')$ each edge of which simulates the R -part of one edge of A and then the L -part of another edge of A . Hence, in A' , every edge is of the form (p, x, RL, q) with $p, q \in Q'$, $R \in \text{RAT}(\{a\}^\oplus \times C)$, and $L \in \text{RAT}(\{\bar{a}\}^\oplus \times C)$.

Since $\{a\}^\oplus \times C$ and $\{\bar{a}\}^\oplus \times C$ are commutative, all such R and L are semilinear sets and we can even assume that every edge is of the form $(p, x, R^\oplus(m, c)L^\oplus, q)$, in which $(m, c) \in \mathbb{B} \times C$ and R and L are finite subsets of $\{a\}^\oplus \times C$ and $\{\bar{a}\}^\oplus \times C$, respectively.

Now the first crucial observation is that if we allow the transducer to apply elements of $\{a\}^\oplus \times C$ that, in an edge $(p, x, R^\oplus(m, c)L^\oplus, q)$ traversed earlier, were contained in R , we do not increase the set of accepted pairs in $X^* \times C$. This is due to the fact that if the counter realized by \mathbb{B} does not go below zero in this new computation, it will certainly not go below zero if we add the value at hand in an earlier step. Thus, any computation in the new transducer can be transformed into one in the old transducer. Furthermore, the commutativity of C guarantees that the output is invariant under this transformation. Analogously, if we allow the transducer to apply elements from $\{\bar{a}\}^\oplus \times C$, as long as it ensures that in some edge $(p, x, R^\oplus(m, c)L^\oplus, q)$ traversed later, they are contained in L , we do not change the accepted set of pairs either.

Therefore, we construct a rationally labeled transducer A'' from A' . In its state, A'' stores a state of A' and two sets: a finite set $\tilde{R} \subseteq \{a\}^\oplus \times C$ and a finite set $\tilde{L} \subseteq \{\bar{a}\}^\oplus \times C$. \tilde{R} always contains all those elements of $\{a\}^\oplus \times C$ that have occurred in sets R so far, and \tilde{L} are elements of $\{\bar{a}\}^\oplus \times C$ that still have to be encountered in sets L in the future. Then for every edge $(p, x, R^\oplus(m, c)L^\oplus, q)$ in A' , we have an edge labeled $(x, (R \cup \tilde{R})^\oplus(m, c)(L \cup \tilde{L})^\oplus)$. A'' will then add the elements of R to its set \tilde{R} and nondeterministically remove some elements of L from \tilde{L} (they can only be removed if this is their last occurrence; otherwise, we might need them in \tilde{L} later). The final state will then make sure that \tilde{L} is empty and A'' has thus only applied elements early that would later appear. In the initial state, both sets \tilde{R} and \tilde{L} are empty and then \tilde{L} is filled nondeterministically.

We have constructed A'' to have the following property. For every computation

$$q_0 \xrightarrow{(x_1, R_1^\oplus(m_1, c_1)L_1^\oplus)} q_1 \cdots q_{n-1} \xrightarrow{(x_n, R_n^\oplus(m_n, c_n)L_n^\oplus)} q_n,$$

we have

$$R_1 \subseteq R_2 \subseteq \cdots \subseteq R_n \quad \text{and} \quad L_1 \supseteq L_2 \supseteq \cdots \supseteq L_n.$$

The essential idea of the proof is that in A'' , we can accept any pair of $X^* \times C$ by a computation such that the element in the R^\oplus -part and the element in the L^\oplus -part in

each edge differ in length (as measured by the number of a and \bar{a}) only by a bounded number. If they differ by more than the bound, either some part of the R^\oplus -part or some part of the L^\oplus -part can be postponed or applied earlier, respectively.

If then we know that these lengths differ only by a bounded number, we will see that for each occurring difference (between the lengths), the set of possible outputs is semilinear. Thus, we only have to output this semilinear set and add this difference.

The valence transducer \hat{A} is obtained from A'' as follows. Let $e = (p, x, R^\oplus(a^k \bar{a}^n, c) L^\oplus, q)$ be an edge in A'' . Let Y and Z be alphabets in bijection with R and L , respectively, and let $\varphi : (Y \cup Z)^\oplus \rightarrow \mathbb{B} \times C$ be the morphism extending these bijections. Furthermore, if $\kappa : \mathbb{B} \rightarrow \mathbb{Z}$ is the morphism with $\kappa(a) = 1$ and $\kappa(\bar{a}) = -1$, let $\psi : (Y \cup Z)^\oplus \rightarrow \mathbb{Z}$ be defined by $\psi(\mu) = \kappa(\pi_1(\varphi(\mu)))$. The set $C_i = \pi_2(\varphi(\psi^{-1}(i))) \subseteq C$ now contains all outputs $c_1 c_2 \in C$ such that there are $(a^t, c_1) \in R^\oplus$ and $(\bar{a}^u, c_2) \in L^\oplus$ with $t - u = i$. Moreover, by Lemma 4.1, the set C_i is semilinear. Let

$$b = \min\{-1, \psi(z) + n - k \mid z \in Z\}, \quad B = \max\{1, \psi(y) + n - k \mid y \in Y\}. \quad (9)$$

\hat{A} has the same set of states as A'' . To simulate the edge e , we introduce for each $i \in \mathbb{N}$ with $b < i < B$ the edge

$$(p, x, a^{k+i} \bar{a}^n, c C_i, q) \quad \text{if } i \geq 0, \quad (10)$$

$$(p, x, a^k \bar{a}^{n-i}, c C_i, q) \quad \text{if } i < 0. \quad (11)$$

Initial state and final states remain unaltered. We claim that $\Phi(T(\hat{A})) = T(A'')$. By the construction, it is clear that $\Phi(T(\hat{A})) \subseteq T(A'')$. Now consider a computation in A'' with steps $p \xrightarrow{(x, r(a^k \bar{a}^n, c) \ell)} q$ for edges $(p, x, R^\oplus(a^k \bar{a}^n, c) L^\oplus, q)$. Define $\varphi, \kappa, \psi, m, M$ as above. Let $r = \varphi(\mu)$ and $\ell = \varphi(\nu)$, $\mu \in Y^\oplus$, $\nu \in Z^\oplus$.

Suppose there is a y in μ such that

$$\psi(\mu - y) + k - n + \psi(\nu) \geq 0, \quad (12)$$

that is, the counter stays above zero until the end of the step, even if we do not add y . Then the counter will also stay above zero if we postpone the application of $\varphi(y)$ until the beginning of the next step. By construction, A'' allows us to do so. Note that we cannot be in the last step of the computation, since this would leave a positive value on the counter. Analogously, suppose there is a z in ν such that

$$-\psi(\nu - z) + n - k - \psi(\mu) \geq 0, \quad (13)$$

that is, when starting from the right (and interpreting \bar{a} as increment and a as decrement), the counter does not drop below zero until the beginning of the step, even if we do not apply $\varphi(z)$. Then we can apply $\varphi(z)$ earlier in the computation. Again, note that this cannot happen in the first step, since this would mean the computation starts by subtracting from the counter.

We transform the computation in the following way. Whenever in some step, (12) is satisfied, we move $\varphi(y)$ to the right (i.e., we postpone the application of $\varphi(y)$). Symmetrically, whenever in some step, (13) is fulfilled, we move $\varphi(z)$ to the left (i.e., we

apply $\varphi(z)$ earlier). We repeat this and since the computation is finite, this process will terminate and we are left with a valid equivalent computation in which (12) and (13) do not occur.

The equations (12) and (13) are equivalent to

$$\begin{aligned}\psi(\mu) + \psi(\nu) &\geq \psi(y) + n - k, \\ \psi(\mu) + \psi(\nu) &\leq \psi(z) + n - k.\end{aligned}$$

Since these are not satisfied, we have

$$\psi(\mu) + \psi(\nu) < \psi(y) + n - k \quad \text{for each } y \text{ in } \mu, \quad (14)$$

$$\psi(\mu) + \psi(\nu) > \psi(z) + n - k \quad \text{for each } z \text{ in } \nu \quad (15)$$

and thus

$$b < \psi(\mu) + \psi(\nu) < B.$$

Note that these inequalities follow from (15) and (14), respectively, if $\nu \neq 0$ and $\mu \neq 0$. In case $\mu = 0$ or $\nu = 0$, they still hold because then $\psi(\mu) + \psi(\nu)$ is ≤ 0 or ≥ 0 , respectively, and $b < 0$ and $0 < B$. This means, however, that each step has a counterpart in the edges (10) and (11). Therefore, $\Phi(T(\hat{A})) = T(A'')$. \square

Lemma 7.7. *Suppose $M \in \mathcal{C}$ is strongly λ -independent. Then $M \times \mathbb{Z}$ is strongly λ -independent as well.*

Proof. In order to simplify notation, we write the operation of C with $+$. Let $T \in \mathbf{VT}^\lambda(M \times \mathbb{Z}, C)$ and let $A = (Q, X, M \times \mathbb{Z}, C, E, q_0, F)$ be a transducer for T . By letting $E' = \{(p, x, m, (z, c), q) \mid (p, x, (m, z), c, q) \in E\}$, we get a transducer $A' = (Q, X, M, \mathbb{Z} \times C, E', q_0, F)$. Then we have $(w, c) \in T$ if and only if there is a $(w, (0, c)) \in T(A')$. By the hypothesis, there is a λ -free valence transducer A'' over M with output in $\mathbf{SL}(\mathbb{Z} \times C)$ such that $\Phi(T(A'')) = T(A')$.

In A'' , every edge is of the form (p, x, m, S, q) , where $S \subseteq \mathbb{Z} \times C$ is semilinear. Thus, we can assume that every edge is of the form $(p, x, m, (\ell, c) + S^\oplus, q)$, where $S \subseteq \mathbb{Z} \times C$ is finite. Since $\mathbb{Z} \times C$ is commutative, we do not change the transduction if we output elements $s \in \mathbb{Z} \times C$ that occur in some S in a step anywhere else in the computation. Therefore, we can transform A'' so as to make it guess the set \tilde{S} of all $s \in \mathbb{Z} \times C$ that will occur in an S somewhere in the computation. It uses its finite control to guarantee that the computation is only accepting if all elements of \tilde{S} actually occur. In every step, it allows the output of every element of \tilde{S}^\oplus . Thus, in the resulting transducer A''' , we have that in any computation, the set S in steps $p \xrightarrow{(x, m, (\ell, c) + S^\oplus)} q$ does not change throughout the computation.

Our new transducer \hat{A} has the same set of states as A''' and the edges are defined as follows. For the edge $(p, x, m, (\ell, c) + S^\oplus, q)$ in A''' , let Y be an alphabet in bijection with S and let $\varphi : Y^\oplus \rightarrow \mathbb{Z} \times C$ be the morphism extending this bijection. Let $k \in \mathbb{N}$ be the constant provided by Lemma 4.1 for the map $\psi : Y^\oplus \rightarrow \mathbb{Z}$, $\psi(\mu) = \pi_1(\varphi(\mu))$. We introduce an edge

$$(p, x, (m, \ell + \psi(\nu)), c + \pi_2(\varphi(\nu + \ker \psi)), q)$$

for every $\nu \in Y^\oplus$ with $|\nu| \leq k \cdot B$, where B is the maximum over all values $|\ell|$ for edges $(p', x', m', (\ell, c') + S^\oplus, q')$ in A''' . Initial and final states remain unaltered. Note that by Lemma 4.1, the set $c + \pi_2(\varphi(\nu + \ker \psi)) \subseteq C$ is semilinear. Observe that each of these edges chooses an element of S^\oplus , namely a $\mu \in \nu + \ker \psi$, and adds $\ell + \psi(\nu) = \ell + \psi(\mu)$ to the \mathbb{Z} -component of the storage and outputs $c + \pi_2(\varphi(\mu))$. Thus, it simulates a step in A''' . Therefore, if $(w, c) \in \Phi(T(\hat{A}))$, then $(w, (0, c)) \in \Phi(T(A'''))$ and thus $(w, c) \in T$.

It remains to be shown that $(w, (0, c)) \in \Phi(T(A'''))$ implies $(w, c) \in \Phi(T(\hat{A}))$. Therefore, Let

$$q_0 \xrightarrow{(x_1, m_1, (\ell_1, c_1) + s_1)} q_1 \cdots q_{n-1} \xrightarrow{(x_n, m_n, (\ell_n, c_n) + s_n)} q_n$$

be a computation in A''' that witnesses $(w, (0, c)) \in \Phi(T(A'''))$. Let $s_i \in S^\oplus$ for $1 \leq i \leq n$ and define Y, φ, ψ, k, B as above. Let $s_i = \varphi(\mu_i)$, $\mu_i \in Y^\oplus$. Since the computation accepts $(w, (0, c))$, we have $\psi(\mu_1 + \cdots + \mu_n) + \ell_1 + \cdots + \ell_n = 0$ and for $\mu = \mu_1 + \cdots + \mu_n$ we have thus

$$|\psi(\mu)| = |\psi(\mu_1 + \cdots + \mu_n)| = |\ell_1 + \cdots + \ell_n| \leq n \cdot B.$$

Lemma 4.1 now yields a $\nu \sqsubseteq \mu$ with $\mu \in \nu + \ker \psi$ and $|\nu| \leq knB$. This means that we can write $\nu = \nu_1 + \cdots + \nu_n$ such that $|\nu_i| \leq kB$ for $1 \leq i \leq n$. Since $\mu \in \nu + \ker \psi$, we have $s_i \in \varphi(\nu + \ker \psi)$ and

$$\ell_1 + \psi(\nu_1) + \cdots + \ell_n + \psi(\nu_n) = \ell_1 + \cdots + \ell_n + \psi(\mu) = 0.$$

Thus, using the edges

$$q_0 \xrightarrow{(x_1, (m_1, \ell_1 + \psi(\nu_1)), c_1 + \pi_2(\varphi(\nu_1 + \ker \psi)))} q_1 \cdots q_{n-1} \xrightarrow{(x_n, (m_n, \ell_n + \psi(\nu_n)), c_n + \pi_2(\varphi(\nu_n + \ker \psi)))} q_n,$$

we have $(w, c) \in \Phi(T(\hat{A}))$. □

Lemma 7.8. *If $\varphi : M \rightarrow N$ is a morphism with $\varphi^{-1}(1) = \{1\}$, then $\text{VT}(M, C) \subseteq \text{VT}(N, C)$ for any monoid C .*

Proof. Take a valence transducer over M with output in C and replace each edge (p, x, m, c, q) by $(p, x, \varphi(m), c, q)$. This yields a valence transducer over N that performs the same transduction. □

Lemma 7.9. *Suppose $M \in \mathcal{C}$ is non-trivial and strongly λ -independent. Then $M * \mathbb{B}$ is strongly λ -independent as well.*

Proof. By Lemma 7.5, in order to show $T \in \Phi(\text{VT}(M * \mathbb{B}, \text{SL}(C)))$ for any given $T \in \text{VT}^\lambda(M * \mathbb{B}, C)$, we can assume that $T = T(A)$ for a rationally labeled valence transducer A over $M * \mathbb{B}$ with output in C . Without loss of generality, we can assume that in $A = (Q, X, M * \mathbb{B}, C, E, q_0, F)$, we have $E \subseteq Q \times X \times \text{RAT}((M * \mathbb{B}) \times C) \times Q$ and $F = \{q_f\}$.

First, we claim that $\text{VT}(M * \mathbb{B}^{(n)}, C) = \text{VT}(M * \mathbb{B}, C)$ for any commutative C . In fact, since $M \in \mathcal{C}$ is non-trivial, it contains an element $b \in R(M)$ such that $b^i \neq b^j$ for $i \neq j$, $i, j \in \mathbb{N}$. Let $b\bar{b} = 1$ and let $a_1, \bar{a}_1, \dots, a_n, \bar{a}_n$ be the generators of the factors \mathbb{B} in $M * \mathbb{B}^{(n)}$, respectively. Then the map $\varphi : M * \mathbb{B}^{(n)} \rightarrow M * \mathbb{B}$, with $\varphi(a_i) = ab^i a$,

$\varphi(\bar{a}_i) = \bar{a}\bar{b}^i\bar{a}$, $\varphi(m) = m$, for $1 \leq i \leq n$ and $m \in M$, clearly satisfies $\varphi^{-1}(1) = \{1\}$. Thus, by Lemma 7.8, we have $\text{VT}(M * \mathbb{B}^{(n)}, C) = \text{VT}(M * \mathbb{B}, C)$ and it will suffice to show

$$T(A) \in \Phi(\text{VT}(M * \mathbb{B}^{(n)}, \text{SL}(C)))$$

for some $n \in \mathbb{N}$.

By Corollary 6.3, we can assume that for every edge $(p, (x, S), q) \in E$, there is an alphabet $X' = \{x, \bar{x}\} \cup Y \cup Z$, a morphism $\varphi : X'^* \rightarrow (M * \mathbb{B}) \times C$ with $\varphi(x) = a$, $\varphi(\bar{x}) = \bar{a}$, $\varphi(Y) \subseteq M$, $\varphi(Z) \subseteq C$, and rational languages

$$L \subseteq ((Y \cup Z)^* \bar{x})^*, \quad J \subseteq (Y \cup Z)^*, \quad R \subseteq (x(Y \cup Z)^*)^* \quad (16)$$

such that $S \cap (J(M * \mathbb{B}) \times C) = \varphi(L)\varphi(J)\varphi(R)$. Indeed: for those edges where the union provided by Corollary 6.3 ranges over more than one set, we can introduce new edges. Without loss of generality, we assume that the alphabets X', Y, Z are the same for all edges $(p, (x, S), q)$. If we replace the edge $(p, (x, S), q)$ by $(p, (x, \varphi(LJR)), q)$, we do not change the transduction, since elements outside of $J(M * \mathbb{B})$ cannot occur in a product that results in 1. Therefore, we assume that every edge of A is of the form $(p, (x, \varphi(LJR)), q)$ as in (16).

In order to be able to denote several appearing rational sets using a pair of states, we construct finite automata

$$\begin{aligned} B^{(-)} &= (Q^{(-)}, X', E^{(-)}, q_0, \emptyset), \\ B^{(0)} &= (Q^{(0)}, Y \cup Z, E^{(0)}, q_0, \emptyset), \\ B^{(+)} &= (Q^{(+)}, X', E^{(+)}, q_0, \emptyset) \end{aligned}$$

such that for each edge $(p, (x, \varphi(LJR)), q) \in E$, we have $L = L_{r,s}(B^{(-)})$, $J = L_{t,u}(B^{(0)})$, and $R = L_{v,w}(B^{(+)})$ for some states $r, s \in Q^{(-)}$, $t, u \in Q^{(0)}$, $v, w \in Q^{(+)}$. Because of (16), we can assume that in these automata there are subsets $\tilde{Q}^{(-)} \subseteq Q^{(-)}$, $\tilde{Q}^{(+)} \subseteq Q^{(+)}$ such that in $B^{(-)}$, an edge is labeled \bar{x} if and only if it enters a state in $\tilde{Q}^{(-)}$ and an edge in $B^{(+)}$ is labeled x if and only if it leaves a state in $\tilde{Q}^{(+)}$. For each $r, s \in \tilde{Q}^{(-)}$, $t, u \in Q^{(0)}$, $v, w \in \tilde{Q}^{(+)}$, let

$$\begin{aligned} L_{r,s} &= \varphi(L_{r,s}(B^{(-)})), \quad J_{t,u} = \varphi(L_{t,u}(B^{(0)})), \quad R_{v,w} = \varphi(L_{v,w}(B^{(+)})), \\ \tilde{L}_{r,s} &= \{\varphi(w) \mid w \in (Y \cup Z)^*, w\bar{x} \in L_{r,s}(B^{(-)})\}, \\ \tilde{R}_{v,w} &= \{\varphi(w) \mid w \in (Y \cup Z)^*, xw \in L_{v,w}(B^{(+)})\}. \end{aligned}$$

By (16), every edge in A is of the form $(p, (x, L_{r,s}J_{t,u}R_{v,w}), q)$.

The essential idea of the construction is to maintain a representation of a set of possibly reached configurations. Roughly speaking, we represent a sequence of rational subsets of $R(M * \mathbb{B}) \times C$ by elements of $M * \mathbb{B}^{(n)}$. Now in order to simulate the multiplication of a set of the form $L_{r,s}$, we have to output a set of elements of C that appear as output while canceling out elements on the stack with those in $L_{r,s}$. Therefore, we will output sets of the form

$$C_{v,w,r,s} = \{c \in C \mid (1, c) \in R_{v,w}L_{r,s}\}.$$

By Lemma 4.5, these sets are semilinear.

In the course of a computation, we will have to simulate the multiplication of rational subsets of $M \times C$. To this end, we will use the hypothesis of M being strongly λ -independent in the following way. Assume that

$$W = \tilde{Q}^{(-)} \times \tilde{Q}^{(-)} \cup Q^{(0)} \times Q^{(0)} \cup \tilde{Q}^{(+)} \times \tilde{Q}^{(+)}$$

is an alphabet. Let $D = (\{q\}, W, M, C, E', q, \{q\})$ be the rationally labeled valence transducer over M with output in C with the following edges:

- for each $r, s \in \tilde{Q}^{(-)}$, create a loop on q with input $(r, s) \in W$ and label $\tilde{L}_{r,s}$,
- for each $t, u \in Q^{(0)}$, create a loop on q with input $(t, u) \in W$ and label $J_{t,u}$, and
- for each $v, w \in \tilde{Q}^{(+)}$, create a loop on q with input $(v, w) \in W$ and label $\tilde{R}_{v,w}$.

Since M is strongly λ -independent, we can transform D into a λ -free valence transducer $\hat{D} = (\hat{Q}, X, M, \text{SL}(C), \hat{E}, q_0, \hat{F})$ over M with output in $\text{SL}(C)$ such that $\Phi(T(\hat{D})) = T(D)$.

As mentioned above, we will encode rational subsets of $\text{R}(M * \mathbb{B}) \times C$ by elements of $M * \mathbb{B}^{(n)}$. The monoid structure of $M * \mathbb{B}^{(n)}$ allows us to use the positive generators of the n instances of \mathbb{B} as stack symbols. Specifically, for every pair of states $v, w \in \tilde{Q}^{(+)}$, we will have a symbol $\Delta_{v,w}^\circ$ that represents the set $R_{v,w}$.

First suppose that sets $J_{t,u}$ do not occur on edges. When an element of $R_{v,w}$ is completely canceled out by an element of $L_{r,s}$, then we have to output an element of $C_{v,w,r,s}$. Therefore, when $\Delta_{v,w}^\circ$ is on top and we want to simulate the multiplication of $L_{r,s}$, we output the semilinear set $C_{v,w,r,s}$ and remove $\Delta_{v,w}^\circ$.

By construction, composing an element of $R_{v,w}$ with one of $L_{r,s}$ always yields one whose first component is in some $\pi_1(R_{v,w'})$, $w' \in \tilde{Q}^{(+)}$ or one outside of $\text{J}(M * \mathbb{B}) \times C$. Therefore, in order to simulate a computation where an element of $L_{r,s}$ cancels out only part of an element of $R_{v,w}$, we have a *split* operation, which removes a symbol $\Delta_{v,w}^\circ$ from the top and puts $\Delta_{v,w'}^\circ \Delta_{w',w}^\circ$ in its place, so that the simulation of $L_{r,s}$ can then cancel out $\Delta_{w',w}^\circ$ and output $C_{w',w,r,s}$. Note that the set $C_{w',w,r,s}$ contains only the outputs for those compositions where the elements actually cancel out. In particular, those compositions that yield elements outside of $\text{J}(M * \mathbb{B}) \times C$ provide no output in $C_{w',w,r,s}$.

In order to simulate an element of $L_{r,s}$ that cancels out an element in the composition of the two topmost rational sets, we need a way to merge two representations of rational sets. However, if we would merge two representations of rational subsets of $\text{R}(M * \mathbb{B}) \times C$ into one, the resulting representation would not be of the form $\Delta_{v,w}^\circ$, since it has to keep track of what states have to be visited on the way. Furthermore, the more representations we would merge, the more information we would have to maintain.

Therefore, we will not merge representations of the form $\Delta_{v,w}^\circ$. Instead, we have another kind of symbols: the symbol $\Delta_{r,s}^\bullet$ stands for an element of $\text{R}(M * \mathbb{B}) \times C$ that can be canceled out by one of $L_{r,s}$. Furthermore, the occurrence of such a symbol also implies that the corresponding output of the canceling process has already been performed. This

means, the symbol $\Delta_{r,s}^\bullet$ is produced by an operation *cancel* that removes $\Delta_{v,w}^\circ$, places $\Delta_{r,s}^\bullet$ on top and outputs $C_{v,w,r,s}$. Since C is commutative, this early output does not change the result. The *merge* operation then consists of removing $\Delta_{r,s}^\bullet \Delta_{s,s'}^\bullet$ and putting $\Delta_{r,s'}^\bullet$ in its place. Since we will always be able to assume that a symbol $\Delta_{v,w}^\circ$ has already been turned into a $\Delta_{r,s}^\bullet$, we can always simulate the application of a set $L_{r,s}$ by removing $\Delta_{r,s}^\bullet$.

Finally, we have to simulate the application of sets $J_{t,u}$. To this end, we use an edge $(p, (t, u), m, S, q)$ in the transducer \hat{D} . The state information of \hat{D} is then stored in symbols \square_p on the stack. Thus, we simulate $J_{t,u}$ by removing \square_p from the stack, using S as output, and adding $m\square_q$ on the stack.

In order to let elements of M that are factors of elements in $R_{v,w}$, i.e., elements of $\tilde{R}_{v,w}$, interact with sets $J_{r,s}$, we have two further operations: *convert-to* and *convert-from*. *Convert-to- M* removes an element $\Delta_{v,w}^\circ$ from the stack and instead adds $\square m \square_q$ on the stack and outputs S , where $(q_0, (v, w), m, S, q)$ is an edge in \hat{D} . That is, the element represented by $\Delta_{v,w}^\circ$ can be thought of as being handed over to \hat{D} . Here, \square represents the a that was part of $R_{v,w}$, but not of $\tilde{R}_{v,w}$. Thus, *convert-from- M* initiates a subsequence of stack elements that simulate a computation of \hat{D} . On the other hand, *convert-from- M* will terminate such a subsequence by simulating the multiplication of a set of the form $\tilde{L}_{r,s}$. It removes \square_q , adds m , removes \square , adds $\Delta_{r,s}^\bullet$, and outputs S , where $(q, (r, s), m, S, q_f)$ is an edge in \hat{D} and q_f is a final state of \hat{D} .

Formally, let Θ be the alphabet

$$\Theta = \{\Delta_{v,w}^\circ, \Delta_{r,s}^\bullet, \square_q, \square \mid v, w \in \tilde{Q}^{(+)}, r, s \in \tilde{Q}^{(-)}, q \in \hat{Q}\}$$

and let $n = |\Theta|$. We let each of the symbols $x \in \Theta$, together with its counterpart \bar{x} , be the generators of one of the instances of \mathbb{B} in $M * \mathbb{B}^{(n)}$. Sometimes, it is necessary to apply one of the aforementioned operations not on top of the stack, but on one symbol below the top. Therefore, for the operations *split*, *merge*, and *cancel*, we have a *deep* variant, which nondeterministically removes some $x \in \Theta$, then performs the original operation and then puts x back on top.

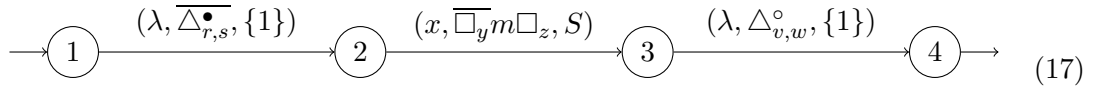
Formally, an operation is a (finite) set of elements of $(X \cup \{\lambda\}) \times (M * \mathbb{B}^{(n)}) \times \text{SL}(C)$. In accordance with the explanation above, we have the following operations:

- $\text{split} = \{(\lambda, \overline{\Delta_{v,w}^\circ} \Delta_{v,v'}^\circ \Delta_{v',w}^\circ, \{1\}) \mid v, v', w \in \tilde{Q}^{(+)}\}$
- $\text{deep-split} = \{(\lambda, \bar{x}sx, S) \mid x \in \Theta, (\lambda, s, S) \in \text{split}\}$
- $\text{merge} = \{(\lambda, \overline{\Delta_{r,r'}^\bullet \Delta_{r',s}^\bullet} \Delta_{r,s}^\bullet, \{1\}) \mid r, r', s \in \tilde{Q}^{(-)}\}$
- $\text{deep-merge} = \{(\lambda, \bar{x}sx, S) \mid x \in \Theta, (\lambda, s, S) \in \text{merge}\}$
- $\text{deep-deep-merge} = \{(\lambda, \bar{x}sx, S) \mid x \in \Theta, (\lambda, s, S) \in \text{deep-merge}\}$
- $\text{convert-to} = \{(\lambda, \overline{\Delta_{v,w}^\circ} \square m \square_q, S) \mid v, w \in \tilde{Q}^{(+)}, (q_0, (v, w), m, S, q) \in \hat{E}\}$
- $\text{convert-from} = \{(\lambda, \overline{\square_q m \square} \Delta_{r,s}^\bullet, S) \mid r, s \in \tilde{Q}^{(-)}, (q, (r, s), m, S, q_f) \in \hat{E}, q_f \in \hat{F}\}$

- $\text{deep-convert-from} = \{(\lambda, \bar{x}sx, S) \mid x \in \Theta, (\lambda, s, S) \in \text{convert-from}\}$
- $\text{cancel} = \{(\lambda, \overline{\Delta_{v,w}^\circ} \Delta_{r,s}^\bullet, C_{v,w,r,s}) \mid v, w \in \tilde{Q}^{(+)}, r, s \in \tilde{Q}^{(-)}\}$
- $\text{deep-cancel} = \{(\lambda, \bar{x}sx, S) \mid x \in \Theta, (\lambda, s, S) \in \text{cancel}\}$
- $\text{deep-deep-cancel} = \{(\lambda, \bar{x}sx, S) \mid x \in \Theta, (\lambda, s, S) \in \text{deep-cancel}\}$

We will now describe the transducer \hat{A} in detail. Although \hat{A} will have λ -transitions, we will argue later that every element of $T(A)$ can be accepted by \hat{A} using a computation that uses only a bounded number of λ -transitions before and after every non- λ -transition. Thus, it is clearly possible to transform \hat{A} into an equivalent λ -free valence transducer over $M * \mathbb{B}^{(n)}$.

\hat{A} is obtained from A by removing all edges and then for each edge $(p, x, L_{r,s}J_{t,u}R_{v,w}, q)$, gluing in the automaton



between p and q for every edge $(y, (t, u), m, S, z)$ in \hat{D} . Furthermore, on every state of \hat{A} (including those in the glued in automata), we add loops labeled with the operations defined above. Finally, we add a loop labeled $(\lambda, \Delta_{r,r}^\bullet, \{1\})$ for each $r \in \tilde{Q}^{(-)}$ on the initial state and a loop labeled $(\lambda, \overline{\Delta_{v,v}^\circ}, \{1\})$ for each $v \in \tilde{Q}^{(+)}$ on each final state.

By the definition, it is clear that $\Phi(T(\hat{A})) \subseteq T(A)$. On the other hand, we can accept every pair $(w, c) \in T(A)$ by \hat{A} in the following way. First we bring a symbol $\Delta_{r,r}^\bullet$ on the stack to represent an empty storage. The first simulated edge $(p, x, L_{r,s}J_{t,u}R_{v,w}, q)$, which has $s = r$, will thus be able to take the edge from state 1 to 2 in 17. We assume that on the stack, there are no symbols of the form $\Delta_{r,s}^\bullet$ except for one representing the empty stack. Thus, the monoid element in the configuration is contained in

$$\{\Delta_{v,w}^\circ, \square m \square_y \mid v, w \in \tilde{Q}^{(+)}, m \in M, y \in \hat{Q}\}^* \cup \{\Delta_{r,r}^\bullet \mid r \in \tilde{Q}^{(-)}\}.$$

For each edge $(p, x, L_{r,s}J_{t,u}R_{v,w}, q)$ in the computation in A :

1. Apply a sequence of **cancel/deep-cancel**, **convert-from/deep-convert-from**, **merge**, and **split/deep-split** loops in state 1 to obtain the symbol $\Delta_{r,s}^\bullet$ on top of the stack. Note that for this, we need to use a **split** or **deep-split** loop at most once, namely for the lowest used occurrence of a $\Delta_{v',w'}^\circ$, which might be canceled only partially.
2. Use the edge $(\lambda, \overline{\Delta_{r,s}^\bullet}, \{1\})$ in (17).
3. If necessary, use a **split** loop in state 2.
4. If necessary, use a **convert-to** loop in state 2.
5. Choose an edge $(x, \overline{\square_y} m \square_z, S)$ in (17).
6. Use the edge $(\lambda, \Delta_{v,w}^\circ, \{1\})$ in (17).

Note that the only one of these phases which uses an unbounded number of operations is the first one. Therefore, we will change the computation by moving the operations of this phase to the point where the modified symbols are created. Thereby, we guarantee that before the application of $(\lambda, \overline{\Delta_{r,s}^\bullet}, \{1\})$ we only need a bounded number of steps. This is done as follows. In each of the phases 2 through 6:

- (i) After each introduction of a $\Delta_{v,w}^\circ$ (by a **split/deep-split** or by adding $\Delta_{v,w}^\circ$ directly): if this occurrence is eventually canceled (without being split), cancel it now. This can be done using **deep-cancel** or **deep-deep-cancel**.
- (ii) After each application of a $\overline{\square_y m \square_z}$: if the corresponding subsequence $\square m' \square_z$ is eventually converted (without adding another $\overline{\square_z m'' \square_{z'}}$), convert it now. This can be done using **convert-from**.
- (iii) Whenever a symbol $\Delta_{r,s}^\bullet$ produced by (i) or (ii) is eventually merged with a symbol below it, merge them now. This can be done using **merge**, **deep-merge**, or **deep-deep-merge**.

If we also obey these rules in the in the first phase, we can assume that any $\Delta_{v,w}^\circ$ or $\square m \square_z$ -subsequence that was canceled/converted and then merged with the current underlying symbol in the old computation, is now already merged. Therefore, we can change the first phase so as to do only a bounded number of operations to obtain $\Delta_{r,s}^\bullet$ on top of the stack. Note that obeying rules (i) through (iii) will not yield an unbounded number of operations, since in the first phase, there is at most one occurrence of **split** or **deep-split**.

In the end, the stack should contain a symbol $\Delta_{v,v}^\circ$, $v \in \tilde{Q}^{(+)}$, to represent the empty storage. This can then be removed by the loop labeled $(\lambda, \overline{\Delta_{v,v}^\circ}, \{1\})$ on the final state.

Thus, any $(w, c) \in T(A)$ can be produced by a computation in \hat{A} using only a bounded number of λ -transitions before and after any input symbol. Hence, \hat{A} can be easily transformed into an equivalent valence transducer with no λ -transitions. \square

We are now ready to prove the first main result.

Proof of Theorem 3.1. Clearly, (2) and (3) each imply (4). Lemma 5.5 shows that (4) implies (5). By Lemma 7.1, 7.6, 7.7, and 7.9, (5) implies that $\mathbb{M}\Gamma$ is strongly λ -independent. By Lemma 7.3, this implies (1). Finally, by Lemma 5.4, (1) implies (2) and (3). \square

We will now prove Theorem 3.3. By Theorem 3.1, we already know that when $r \leq 1$, we have $\text{VA}(\mathbb{M}\Gamma) = \text{VA}^\lambda(\mathbb{M}\Gamma)$. Hence, we only have to show that $\text{VA}(\mathbb{M}\Gamma) \subsetneq \text{VA}^\lambda(\mathbb{M}\Gamma)$ if $r \geq 2$. Greibach [12] and, independently, Jantzen [16, 17] have shown that the language

$$L_1 = \{wc^n \mid w \in \{0,1\}^*, n \leq \text{bin}(w)\},$$

can be accepted by a partially blind counter machine with two counters, but not without λ -transitions. Here, $\text{bin}(w)$ denotes the number obtained by interpreting w as a base 2 representation:

$$\text{bin}(w1) = 2 \cdot \text{bin}(w) + 1, \quad \text{bin}(w0) = 2 \cdot \text{bin}(w), \quad \text{bin}(\lambda) = 0.$$

Since we have to show $\text{VA}(\mathbb{B}^r \times \mathbb{Z}^s) \subsetneq \text{VA}^\lambda(\mathbb{B}^r \times \mathbb{Z}^s)$ and we know $L_1 \in \text{VA}^\lambda(\mathbb{B}^r \times \mathbb{Z}^s)$, it suffices to prove $L_1 \notin \text{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$. We do this by transforming Greibach's proof into a general property of languages accepted by valence automata without λ -transitions. We will then apply this to show that $L_1 \notin \text{VA}(\mathbb{B}^r \times \mathbb{Z}^s)$.

Definition 7.10. Let M be a monoid. For $x, y \in M$, write $x \equiv y$ iff x and y have the same set of right inverses. For a finite subset $S \subseteq M$ and $n \in \mathbb{N}$, let $f_{M,S}(n)$ be the number of equivalence classes of \equiv in $S^n \cap R(M)$.

The following notion is also used as a tool to prove lower bounds in state complexity of finite automata [10]. Here, we use it to prove lower bounds on the number of configurations that an automaton must be able to reach in order to accept a language L .

Definition 7.11. Let $n \in \mathbb{N}$. An n -fooling set for a language $L \subseteq \Theta^*$ is a set $F \subseteq \Theta^n \times \Theta^*$ such that

- for each $(u, v) \in F$, we have $uv \in L$, and
- for $(u_1, v_1), (u_2, v_2) \in F$ such that $u_1 \neq u_2$, we have $u_1v_2 \notin L$ or $u_2v_1 \notin L$.

The function $g_L : \mathbb{N} \rightarrow \mathbb{N}$ is defined as

$$g_L(n) = \max\{|F| \mid F \text{ is an } n\text{-fooling set for } L\}.$$

Lemma 7.12. Let M be a monoid and $L \in \text{VA}(M)$. Then there is a constant $k \in \mathbb{N}$ and a finite set $S \subseteq M$ such that $g_L(n) \leq k \cdot f_{M,S}(n)$ for all $n \in \mathbb{N}$.

Proof. Let k be the number of states in the automaton for L and S be the elements appearing on edges. Suppose $g_L(n) > k \cdot f_{M,S}(n)$ for some n and let $F = \{(u_1, v_1), \dots, (u_m, v_m)\}$ be an n -fooling set for L , with $m > k \cdot f_{M,S}(n)$. Since $u_i v_i \in L$ for $1 \leq i \leq m$, we have an accepting computation for each of these words. Let (q_i, x_i) be the configuration reached in such a computation after reading u_i , $1 \leq i \leq m$. Since the automaton has no λ -transitions, we have $x_i \in S^n$ for any $1 \leq i \leq m$. Moreover, since a final configuration is reachable from (q_i, x_i) , we also have $x_i \in R(M)$. Furthermore, since $m > k \cdot f_{M,S}(n)$, there are indices $i \neq j$ with $q_i = q_j$ and $x_i \equiv x_j$. This means however, that $u_i v_j \in L$ and $u_j v_i \in L$, contradicting the fooling set condition. \square

Lemma 7.13. For $L = L_1$, we have $g_L(n) \geq 2^n$ for any $n \in \mathbb{N}$.

Proof. Let $n \in \mathbb{N}$, and let F consist of all (u, v) such that $u \in \{0, 1\}^n$ and $v = c^{\text{bin}(u)}$. Then, F is an n -fooling set for L with $|F| \geq 2^n$: we have $uv \in L$ for any $(u, v) \in F$. Furthermore, if $u \neq u'$ for $(u, v), (u', v') \in F$, assume $\text{bin}(u) < \text{bin}(u')$. Then $uv' \notin L$. \square

Lemma 7.14. Let $M = \mathbb{B}^r \times \mathbb{Z}^s$ for $r, s \in \mathbb{N}$ and $S \subseteq M$ a finite set. Then $f_{M,S}$ is bounded by a polynomial.

Proof. Every $x \in \mathbb{B}$ can be written uniquely as $x = \bar{a}^k a^\ell$. We define $|x| = \ell$. For $y \in \mathbb{Z}$, we have the usual absolute value $|y|$. Thus, for $z \in \mathbb{B}^r \times \mathbb{Z}^s$ and $z = (x_1, \dots, x_r, y_1, \dots, y_s)$ we can define

$$|z| = \max\{|x_i|, |y_j| \mid 1 \leq i \leq r, 1 \leq j \leq s\}.$$

Let $m = \max\{|x| \mid x \in S\}$. Then, for $z \in S^n \cap R(M)$, we have $|z| \leq m \cdot n$. Since every element in $R(M)$ is of the form $(a^{\ell_1}, \dots, a^{\ell_r}, y_1, \dots, y_s)$, we have

$$f_{M,S}(n) \leq |S^n \cap R(M)| \leq (m \cdot n + 1)^r \cdot (2 \cdot m \cdot n + 1)^s,$$

which is a polynomial in n . □

Proof of Theorem 3.3. If $r \leq 1$, Theorem 3.1 already implies that $\text{VA}(\mathbb{M}\Gamma) = \text{VA}^\lambda(\mathbb{M}\Gamma)$. If $r \geq 2$, we have $L_1 \in \text{VA}^\lambda(\mathbb{M}\Gamma)$, but Lemmas 7.12, 7.13, and 7.14 together imply that $L_1 \notin \text{VA}(\mathbb{M}\Gamma)$. □

Acknowledgements The author would like to thank Nils Erik Flick, Reiner Hüchting, Matthias Jantzen, and Klaus Madlener for comments that improved the presentation of the paper.

References

- [1] Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Springer-Verlag New York, Inc., 1993.
- [2] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of push-down automata: Application to model-checking. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997.
- [3] Didier Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290(1):79 – 115, 2003.
- [4] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
- [5] Murray Elder, Mark Kambites, and Gretchen Ostheimer. On groups and counter automata. *International Journal of Algebra and Computation*, 18(08):1345–1364, 2008.
- [6] Gillian Z. Elston and Gretchen Ostheimer. On groups whose word problem is solved by a counter automaton. *Theoretical Computer Science*, 320(23):175–185, 2004.
- [7] Robert H. Gilman. *Formal Languages and Infinite Groups*, volume 25 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. 1996.

- [8] Seymour Ginsburg and Edwin H. Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.
- [9] Seymour Ginsburg and Edwin H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.
- [10] Ian Glaister and Jeffrey Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59(2):75–77, 1996.
- [11] Sheila A. Greibach. A new normal-form theorem for context-free phrase structure grammars. *Journal of the ACM*, 12(1):42–52, 1965.
- [12] Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311 – 324, 1978.
- [13] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.
- [14] Oscar H. Ibarra, Sartaj K. Sahni, and Chul E. Kim. Finite automata with multiplication. *Theoretical Computer Science*, 2(3):271 – 294, 1976.
- [15] Masami Ito, Carlos Martín-Vide, and Victor Mitrană. Group weighted finite transducers. *Acta Informatica*, 38:117–129, 2001.
- [16] Matthias Jantzen. *Eigenschaften von Petrinetzsprachen*. PhD thesis, Universität Hamburg, 1979.
- [17] Matthias Jantzen. Eigenschaften von petrinetzsprachen. Technical Report IFI-HH-B-64/79, Institut für Informatik, Universität Hamburg, 1979.
- [18] Mark Kambites. Formal languages and groups as memory. *Communications in Algebra*, 37:193–208, 2009.
- [19] Markus Lohrey and Benjamin Steinberg. The submonoid and rational subset membership problems for graph groups. *Journal of Algebra*, 320(2):728–755, 2008.
- [20] Victor Mitrană and Ralf Stiebe. Extended finite automata over groups. *Discrete Applied Mathematics*, 108(3):287–300, 2001.
- [21] Elaine Render. *Rational Monoid and Semigroup Automata*. PhD thesis, University of Manchester, 2010.
- [22] Elaine Render and Mark Kambites. Rational subsets of polycyclic monoids and valence automata. *Information and Computation*, 207(11):1329 – 1339, 2009.
- [23] Jan van Leeuwen. A generalisation of parikh’s theorem in formal language theory. In J. Loeckx, editor, *Automata, Languages and Programming*, volume 14 of *Lecture Notes in Computer Science*, pages 17–26. Springer Berlin / Heidelberg, 1974.